

LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
AÑO 3. Nº 21
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
por desarrolladores ESCRITA para desarrolladores
Archivo Edición Ver
CD-ROM DE REGALO
PROGRAMAS Y UTILIDADES PARA LINUX
WWW: EL PROCESAMIENTO DE FORMS
LINUX: EMULADORES DE MS-DOS
UNIX: SERVIDORES CONCURRENTES
Y además:
• LA MEMORIA EXTENDIDA
• REDES LOCALES
• GRANDES SISTEMAS
CURSOS PRÁCTICOS DE:
Formatos de sonido
Creación de un compilador
Visual Basic 4.0
VISUAL TOOLS: AQUÍ Y AHORA
00021
8 413042 303299
TOWER COMMUNICATIONS S.R.L.

SUMARIO

NÚMERO 21

6 **NOTICIAS**
El espacio dedicado a informar al lector de las últimas novedades en el mundo de la informática y el comentario de los libros de interés.

10 **WORLD WIDE WEB**
Se presenta en este artículo el principal método de envío de información a un servidor WWW: el procesamiento de forms.



16 **CURSO DE PROGRAMACIÓN**
En esta entrega se pretende mostrar las distintas fases de análisis y diseño por las que pasa un proyecto para construir un sistema.



22 **CURSO DE UNIX**
En esta ocasión se explica cómo realizar programas de comunicaciones con varios canales, así como entre más de dos procesos.



28 **GRANDES SISTEMAS**
Este mes termina la serie de artículos dedicados a los ficheros MVS explicando una de las más importantes organizaciones, VSAM



33 **CURSO DE DEMOS**
En esta entrega se explica el efecto Shading Bobs, el cual sirve como apoyo para explicar el mundo del coprocesador matemático.



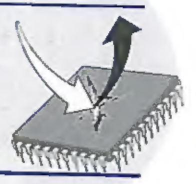
38 **SISTEMA OPERATIVO LINUX**
Este mes se ven los esfuerzos de Linux para aprovechar los programas creados para MS-DOS mediante la utilización de emuladores



44 **MICROSOFT TOOLS**
Continuando con la atención prestada a las herramientas visuales de Microsoft, este mes se incluye un análisis global de todas ellas.



49 CREACIÓN DE UN COMPILADOR
Una vez realizada la definición del lenguaje, en esta ocasión se realiza la introducción a la generación de código.



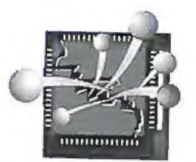
56 REDES LOCALES
Comienza en este número un nuevo curso destinado a explicar al lector el funcionamiento de todo lo relacionado con la redes de área local.



61 FORMATOS DE SONIDO
En esta primera entrega de este nuevo curso se estudian los ficheros que almacenan información musical en forma de eventos y notas.



64 CURSO DE VISUAL BASIC 4.0
Una vez estudiados los formularios, esta nueva entrega explica la manipulación de los distintos controles proporcionados por esta herramienta.



69 VISUAL C++ 4.0
En esta ocasión se analiza la inclusión de todas las nuevas piezas y recursos de lo que compondrán el interfaz de usuario.



75 MEMORIA EXTENDIDA XMS
El estándar XMS está destinado a la gestión de la memoria física situada por encima de los 640 Kbytes de memoria convencional.



79 CONTENIDO DEL CD-ROM
Este mes se incluye un CD en el que podrá encontrarse información sobre numerosos aspectos del sistema operativo Linux



81 CORREO DEL LECTOR
Es el espacio dedicado a la resolución de los problemas surgidos a nuestros lectores en los diversos aspectos de la programación.

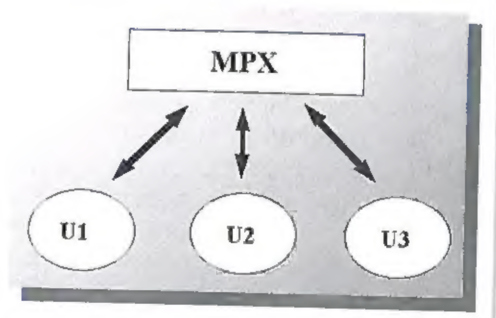


EN PORTADA



Este mes realizamos un completo análisis de las Herramientas Visuales de Microsoft (Microsoft Visual Tools). Algunas de ellas ya poseen su propio espacio dentro de la revista en forma de cursos.

DESTACADO



Tal y como el lector podrá comprobar en la página 27 de la revista, a partir de este mes comenzamos un nuevo concurso sobre la programación de comunicaciones sobre UNIX. Por otro lado, en el número de este mes comienzan nuevos cursos como el de Redes Locales, así como de Formatos de Sonido.

Soluciones de acceso Web a host de Attachmate

Attachmate Corporation ha anunciado recientemente su compromiso de ofrecer soluciones de acceso World Wide Web a host para sus clientes corporativos mediante el producto *Internet Information Server* de Microsoft. Se trata de unas soluciones que combinan las tecnologías *Windows NT Server* e *Internet Information Server* con el producto *Emissary Host Publishing System* de Attachmate y servicio de consultoría. Las nuevas aplicaciones se servidor permiten interactuar con aplicaciones e información residentes en sistemas mainframe y ejecutar cualquier visualizador Web sin necesidad de un software especial.

El producto *Emissary Host Publishing System* se comunica con *Internet Information Server* a través del *API Internet Server* (ISAPI), un interface de programación estándar diseñado específicamente para aplicaciones de servidor Internet. ISAPI ofrece un mayor rendimiento y escalabilidad que el *Common Gateway Interface* (CGI) gracias a la avanzada funcionalidad de Windows NT. Además, el sistema Emissary Host Publishing incorpora un mecanismo SNA, servicios de emulación 3270 integrados y es compatible con SNA Server a través de su soporte a los interfaces de programación de aplicaciones WOSA (*Windows Open Service Architecture*).

Por otro lado Attachmate también ha creado un nuevo grupo de servicios y consultoría para ayudar a sus clientes a crear e implantar nuevas aplicaciones.

Nuevo medio de comunicación para las PYMES europeas

Euredit SA, filial común de las empresas de Telecom europeas ha creado Euronline con el fin de permitir que las pequeñas y medianas empresas europeas se comuniquen a través de la World Wide Web. Euronline proporciona a cada empresa una dirección personal Web con el fin de permitir a los usuarios el acceso a su información. El catálogo o documentación de la empresa son creados a partir de un simple folleto impreso por el departamento editor de Euronline, especializado en la creación de catálogos electrónicos. Dicho departamento también se encarga de las actualizaciones de información y garantiza las traducciones a 5 idiomas.

Euredit inserta los catálogos de las empresas en un conjunto "Business to Business" editorialmente coherente y crea enlaces con los más importantes servicios profesionales de la World Wide Web. También suscribe automáticamente a las empresas en Europages, la guía europea de los negocios con 600.000 ejemplares editados cada año en 30 países de Europa.

El coste de la operación está al alcance de cualquier PYME y los precios se sitúan entre 2.000 y 10.000 FF al año, incluyendo el coste de creación y los gastos técnicos.

Para más información:
Euredit
Tel: +33 (1) 53 77 54 00
URL: http://www.euronline.fr/homepage.html
e-mail: euronlin@europages.com

Cisco Systems adquiere TGV Software

Cisco Systems, Inc. Ha anunciado el acuerdo de adquisición de la compañía TGV Software Inc. (NASDAQ: TGVI), suministrador de productos de software para la interconexión de redes TCP/IP que permiten conectividad entre sistemas heterogéneos sobre redes de área local, extensa y redes informáticas globales. Esta adquisición amplía la línea de productos software, incluyendo aplicaciones y servicios de red, utilizados para construir intranets corporativas de Cisco, y se soporta el crecimiento global de Internet y de la World Wide Web. Además, la línea de gestión, empleados y productos de TGV entran a formar parte de la unidad de negocio Internet de Cisco.

Cisco integrará enlaces de valor añadido entre el software de TGV y el sistema operativo de interconexión Cisco IOS, un sofisticado conjunto de software que asegura robustez y fiabilidad en la interconexión de redes al soportar protocolos LAN y WAN, además de servicios WAN optimizados y control de acceso a redes.

El software TCP/IP MultiNet de TGV ha sido optimizado para entornos UNIX, DEC y Windows, y ofrece a los usuarios un soporte de aplicaciones de valor añadido que incluye transferencia de ficheros, correo electrónico, servicios de impresión, capacidades de copia de seguridad en cinta y reparto de carga (load balancing).

Nuevo conjunto de productos Cisco Advantage

Cisco Systems ha anunciado un conjunto de productos que aumentan la seguridad en Internet, reducen costes administrativos y alivia los cuellos de botella causados por el intenso tráfico en Internet. CiscoAdvantage mejora las capacidades de gestión de red los problemas surgidos a raíz de la creciente popularidad de Internet como la falta de seguridad, los altos volúmenes de tráfico de red y la falta de direcciones de red.

Los productos CiscoAdvantage están diseñados para ser utilizados tanto por proveedores de servicios Internet como por compañías que gestionan sus propias redes. Estos productos aprovechan las capacidades inteligentes de la interconexión de redes para ofrecer servicios más avanzados a los usuarios de redes que quieren acceder al World Wide Web.

A finales del presente año todo el conjunto de productos será HTML-gestionable a través de cualquier software cliente Web como Netscape Navigator o Spyglass Mosaic.

Inicialmente la oferta de productos CiscoAdvantage incluye LocalDirector, DistributedDirector, Private Internet Exchange y el software Internet Junction. El precio europeo de partida para este conjunto de programas es de 8.050 dólares.

Para más información:
Cisco Systems
Avda. De Burgos, 17 planta 11 (Edificio Triada II)
28036 Madrid

Ferias Virtuales en Internet

La empresa valenciana BemarNet Management ofrece un nuevo servicio a empresas en la red mundial de ordenadores Internet. BemarNet ha creado el concepto de las Ferias Virtuales en Internet, que permite a fabricantes y mayoristas ofertar sus productos a través de la red de redes.

Los servicios ofrecidos por BemarNet Management a las empresas expositoras van desde la preparación de las imágenes que serán insertadas en Internet hasta la traducción de todas las páginas Web a cuatro idiomas (inglés, francés, alemán y español). Otros servicios adicionales son la solicitud automática de listas de precios, realización de pedidos en línea (on-line), publicidad de las empresas en Internet y desvío del correo electrónico a fax, entre otros.

BemarNet cuenta con un servidor propio de Internet en España y otro en Estados Unidos. Pronto estará disponible un tercer servidor en Inglaterra y se posibilitará el acceso a Internet de las empresas mediante el servicio infovía de Telefónica.

Para más información:
Tel: +34 6 165 66 44
URL: http://www.bemarnet.es
e-mail: comercial@bemarnet.es

Microsoft adquiere Vermeer Technologies

Microsoft ha anunciado la adquisición de Vermeer Technologies, compañía norteamericana pionera en herramientas visuales de publicación Web basadas en estándares. Vermeer Technologies es la creadora de FrontPage, una reconocida herramienta que simplifica la creación y gestión de documentos Web sin necesidad de programación. Este producto se convertirá en un componente clave de la estrategia de Microsoft destinada a ofrecer una gama completa de productos que incluyan la potencia de la publicación Web tanto para Internet como para intranets corporativas.

FrontPage está diseñado tanto para usuarios individuales como para entornos de trabajo en grupo y su arquitectura cliente/servidor permite la publicación, codificación y gestión de direcciones Web desde un ordenador de sobremesa de usuario, a través de la red de área local de la empresa o sobre Internet. La parte cliente del software dispone de las siguientes características:

- FrontPage Explorer para visualización gráfica y gestión de páginas Web complejas compuestas por múltiples documentos e imágenes.
- Lista de tareas para comprobación del estado o autorización y gestión de tareas a desarrollar en la página.
- Asistentes (Wizards) y plantillas para la creación sencilla de páginas Web personales o corporativas de un modo orientado a tareas.
- FrontPage Editor, para la creación y edición de páginas HTML con soporte

Microsoft anuncia el Service Pack 1 para SNA Server 2.11

Microsoft Ibérica ha anunciado la disponibilidad del Service Pack 1 para SNA Server 2.11. Se trata de una actualización gratuita de mantenimiento para el producto gateway SNA líder en la industria, que se ejecuta sobre Windows NT Server. Este pack de servicio incluye arreglos para todos los problemas descubiertos desde el lanzamiento de SNA Server 2.11, además de un importante número de mejoras a las capacidades actuales y nuevas características que mejoran la integración de SNA Server con redes TCP/IP y que simplifican el acceso a datos residentes en host a través de Internet.

Las características de Service Pack 1 para SNA Server 2.11 incluyen:

- Compatibilidad con los host S/36, S/38 y AS/36 de IBM.
- Drivers para varios adaptadores SDLC y X.25 nuevos.
- Soporte a bases de datos SQL/DS y procedimientos almacenados en el driver ODBC/COBRA.
- Gateway FTP-AFTP.
- Soporte para el enlace a canal ESCON.
- Soporte para clientes TN3270E.
- Servicio de gateway distribuida.
- Soporte al API AFTP.
- Cliente Windows 95 para SNA Server.

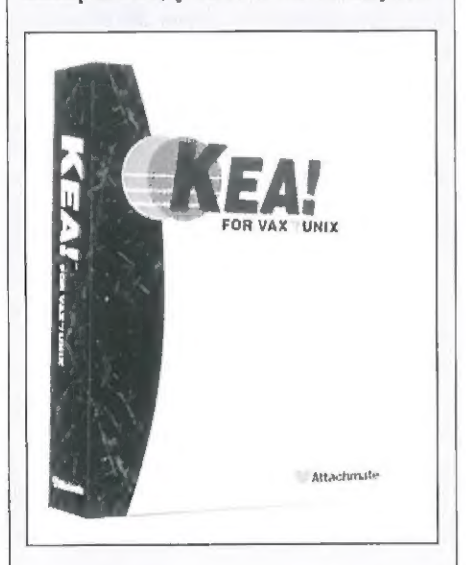
Para más información:
URL: http://www.microsoft.com/sna

NOVEDADES

Nuevas versiones de EXTRA! Y KEA

Attachmate anunció recientemente las nuevas versiones de sus productos EXTRA! Y KEA! dirigidas a satisfacer las necesidades de conectividad UNIX a host de la mayoría de las empresas que utilizan UNIX en sus negocios.

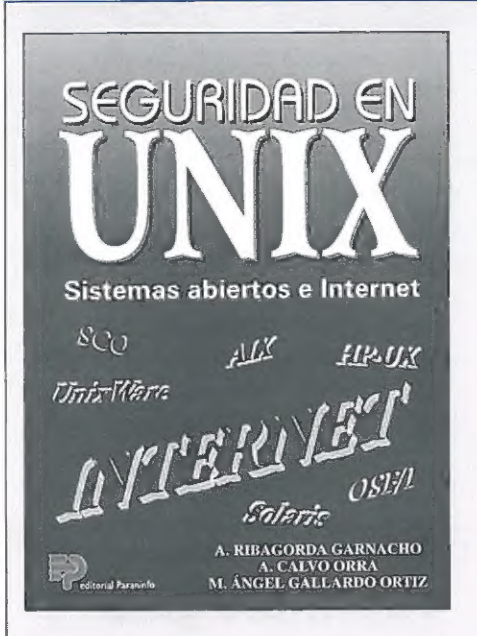
EXTRA! X 2.0 es un potente servidor X para ordenadores PC que integra aplicaciones X con el entono Windows. Se trata de una aplicación de 32 bits diseñada especialmente para ofrecer a los usuarios de Windows 95 y Windows NT un acceso transparente, y también está disponible para Windows 3.1. Este producto incluye nuevas características como mejoras en el rendimiento, el Administrador de aplicaciones X y mapeo gráfico de teclas "arrastrar y soltar". Otras características adicionales son el soporte multi-instancia, el modo multi-visual, el soporte de una amplia variedad de colores que van de 4 a 16 millones, la asociación de fuentes y el soporte completo a las fuentes de Microsoft Windows.



KEA! permite a los usuarios del sis-

LIBROS

Seguridad en UNIX. Sistemas abiertos e Internet



Uno de los aspectos más importantes en la administración de grandes sistemas es la seguridad. Se trata de una obra especializada sobre la seguridad informática en general y la seguridad de los sistemas UNIX en particular. En sus páginas se explican los mecanismos de seguridad, el mantenimiento de sistemas seguros y la programación de aplicaciones seguras en lenguaje C. Además, trata temas más teóricos como los conceptos UNIX relativos a seguridad, el libro naranja, y las instituciones y documentos sobre la seguridad del sistema UNIX en Internet. El libro incluye algunos gráficos ilustrativos y ejemplos.

Editorial: Paraninfo
Autores: A. Ribagorda, A. Calvo y M. Ángel Gallardo
230 páginas
Idioma: Castellano
Precio: Consultar

Estructura de Computadores

Se trata de una obra con fines eminentemente didáticos sobre la estructura, el funcionamiento y la programación de los computadores. El libro está dividido en tres partes bien diferenciadas: teoría, aplicación y prácticas. En la primera se habla sobre la estructura, funcionamiento y programación de los computadores, con temas como las instrucciones, los operadores, la memoria y las entradas-salidas. La segunda parte trata de la estructura y programación del microprocesador 8085, el predecesor de los procesadores de la familia 80x86 de Intel. La última parte está dedicada a simuladores de la Máquina Sencilla, la Máquina+ y el 8085. El libro viene acompañado de un disco con los programas de simulación de tres computadores.



Editorial: Paraninfo
Autor: José María Angulo
589 páginas
Idioma: Castellano
Precio: Consultar

OLE Y LOS FABRICANTES DE HARDWARE

Hasta hace unos años, el mundo del hardware iba unos pasos por delante en relación al del software. Los fabricantes de ordenadores decidieron establecer unos estándares en sus equipos y comprar distintos componentes y microprocesadores de terceras compañías, que luego incorporaban a sus máquinas, ofreciendo configuraciones que respondían a las demandas de los usuarios. Al utilizar estos componentes estándar para todos los equipos, los fabricantes de hardware emplean un menor tiempo y menores costes en la producción de sus máquinas, a la vez que consiguen una mayor flexibilidad. Del mismo modo en el mundo del software y gracias a la tecnología de objetos reutilizables, los desarrolladores de soluciones de software han comenzado a incorporar esta filosofía de trabajo ya utilizada por los fabricantes de hardware: adquirir los objetos que necesiten para sus diseños de terceras compañías y crear sus aplicaciones, disfrutando de las mismas ventajas competitivas que los fabricantes de hardware.

Sin embargo, este planteamiento presenta un problema potencial y es el hecho, de que los desarrolladores obtienen estos objetos de diferentes fabricantes. Es ahí, donde la tecnología OLE tiene su entrada, ofreciendo una garantía de interoperabilidad entre los objetos de software reutilizables de diferentes compañías.

Al utilizar los componentes de OLE, el programador obtiene una gran movilidad entre las aplicaciones, ya que la información definida para una aplicación puede ponerse a disposición de cualquier otra capaz de soportar OLE. Estas aplicaciones incluyen lenguajes de desarrollo como Visual Basic, Visual FoxPro y Visual C++; las aplicaciones de Microsoft Office y otros productos fruto del incremento de terceros fabricantes. Todas estas aplicaciones utilizan la sintaxis común de OLE y pueden acceder a los mismos objetos.

Asimismo, terceros fabricantes y desarrolladores pueden fácilmente añadir nuevos controles de programación a sus entornos de desarrollo, permitiendo así el crecimiento del mercado de controles de terceras partes. De este modo, han surgido en el mercado cientos de controles OLE de 32 bits (OCX), que permiten a los desarrolladores añadir nuevas funcionalidades a sus aplicativos con tan sólo incluir estos controles, que además pueden ser reutilizados a lo largo de un gran número de herramientas de desarrollo, como Visual Basic, Visual C++, Visual FoxPro, Access y otras herramientas de terceras compañías. Microsoft quiere preservar este concepto, al mismo tiempo que ofrece el interfaz estándar OLE, un lenguaje independiente que ofrece soporte para 32 bits. Además, Microsoft presentó sus Controles OLE, un tipo de componentes OLE, para subsanar la falta de un modelo de eventos en esta tecnología; la capacidad de enlace e inclusión de objetos (linking and embedding); la edición visual; la Automatización OLE y la librería de tipos OLE.

La industria del software se puede beneficiar de los componentes de OLE, del mismo modo que la industria del hardware se beneficia de un bus PC o un chip común.

El bus PC, estándar en todos los ordenadores, permite a los fabricantes de hardware crear nuevas e interesantes capacidades y simplemente incorporarlas al PC. En el mismo sentido, el estándar OLE ofrece la garantía de interoperabilidad entre los componentes de software reutilizables.

Cuando un objeto ha sido creado como un estándar, los desarrolladores pueden incluso intercambiarlo con otro de otro fabricante. Por ejemplo, si un objeto de contabilidad no ofrece la funcionalidad necesaria, el objeto puede ser reemplazado por otro de otro fabricante, del mismo modo que un usuario de un PC puede desconectar su modem a 14,4k y conectar otro con mayores capacidades a 28,8k.

Cada vez más los desarrolladores tienen acceso a los componentes y a aplicaciones OLE, que con el tiempo serán más fáciles de desarrollar. Actualmente, los equipos de desarrollo de las compañías pueden crear componentes OLE sin necesidad de amplios conocimientos en desarrollo y pronto, éstos serán tan fáciles de utilizar, como cambiar una tarjeta de red en un PC.

Los desarrolladores deben conocer las capacidades de los distintos tipos de componentes OLE, con el fin de poder utilizar diferentes herramientas de desarrollo para crear sus aplicaciones. Hasta el momento, los entornos de desarrollo estaban limitados a Visual C++ o a Visual Basic, pero actualmente se puede utilizar una combinación de éstos, junto con procesadores de texto, hojas de cálculo y herramientas multimedia. Los desarrolladores pueden configurar los diferentes objetos, así como ofrecer soluciones muy interesantes a los usuarios.

Finalmente, me gustaría añadir los beneficios que la tecnología OLE está ofreciendo ya a los desarrolladores, como: la reutilización de los objetos interoperables; la mejora de la seguridad; la capacidad de adquirir aquellos objetos, disponibles en el mercado, que se adapten mejor a sus necesidades; y la capacidad de desarrollar soluciones reutilizables para el mercado.

José Antonio Alvarez
Jefe de Producto de Herramientas de Desarrollo de Microsoft Ibérica

CGI: ENVÍO DE INFORMACIÓN A UN SERVIDOR

Fernando J. Echevarrieta

Una vez realizada una primera toma de contacto con la interfáz CGI, se presenta en este artículo el principal método de envío de información a un servidor WWW: el procesamiento de *forms*.

En el artículo del mes pasado se realizó una primera toma de contacto con la interfáz CGI y se presentaron las reglas básicas que debía cumplir todo *script*. En el presente artículo se trata la primera extensión que dota al WWW de un sistema de realimentación permitiendo al cliente enviar información al servidor: el uso de *forms*. El procesamiento de *forms* fue uno de los primeros usos que se le dio a la interfáz CGI. Y surgió de la necesidad de enviar información desde el cliente hasta el servidor. El uso más inmediato que se suele dar a los *forms* es la creación de libros de visita para envío de mensajes o, incluso, pasarelas de mail. Sin embargo, como se verá en un ejemplo del artículo, sus funciones pueden ir más allá, sirviendo como ventana para un diálogo o para transmisión de estado oculto a lo largo de una sesión interactiva. El artículo comenzará por una exposición básica seguida de un ejemplo rudimentario. Con la presentación de algunas herramientas se refinará el ejemplo y se procederán a mostrar otras ideas para el uso de *forms*.

PROCESAMIENTO DE FORMS

Como ya se expuso en [Echeva-1], todo *form* consta de dos partes: una página HTML, el *form* propiamente dicho; y un script CGI encargado de procesar la información que esta página recoge. El script deberá respetar todas las reglas que se presentaron el mes anterior pero ahora, además, la interfáz CGI fija la forma en que debe ser codificada la información que proporciona el usuario antes de ser entregada y las posibles formas de hacerla accesible al script. Nuevamente serán irrelevantes el lenguaje de programación y los algoritmos empleados.

El envío de información se sustenta, como todo el sistema, sobre el protocolo HTTP y se puede realizar a través de dos métodos definidos en el mismo: *GET* y *POST*. Recientemente se ha propuesto un draft para el envío de ficheros a servidores WWW como *attachments* a *forms* que hace uso de otro método: *PUT*, aunque por el momento es muy difícil encontrar algún servidor que admita esta posibilidad.

Sin necesidad de adentrarse en el protocolo HTTP, bastará mencionar que será necesario indicar uno de estos dos valores *GET* o *POST* en el atributo *METHOD* de la etiqueta *<FORM>*. Conviene ahora recordar de [Echeva-1] que la etiqueta de declaración de un form tomaba la forma:

<FORM METHOD=método
ACTION=script>

donde script es el URL correspondiente al script CGI encargado de procesar la información del form.

Si el método elegido es *GET*, el servidor creará una variable de entorno para el CGI denominada
QUERY_STRING
e introducirá en ella toda la información del form siguiendo un formato de codificación que se expondrá más adelante.

Si, por el contrario, el método elegido es *POST*, el servidor proporcionará toda la información al CGI a través de la entrada estándar de este último, siguiendo el mismo formato que si se hubiera utilizado el método *GET*.

Como pista para aquellos lectores que no estén acostumbrados a usar variables de entorno en sus programas, se les puede indicar que, para acceder

LISTADO 1

Parser de información procedente de *forms*

```
int CGIparser(char *string, struct pair **p)
/* Extracts pairs {name=value} from string and
return them in list p */
{
	char *token;
	char *name, *value;
	char *auxp1, *auxp2;
	char hex_code[3];
	int  code;

	hex_code[3]='\0';

	while ((token=strsep(&string,"&")) != NULL) {
		name  = strsep(&token,"=");
		auxp1 = strsep(&token,"=");
		value = auxp2 = strdup(auxp1);

		while( *auxp1 != '\0') {
		  switch (*auxp1) {
		    case '+' : *auxp2++ = ' ';
				*auxp1++;
				break;
		    case '%' : *auxp1++;
				hex_code[0] = *auxp1++;
				hex_code[1] = *auxp1++;
				  code=strtol(hex_code,(char
**)NULL,16);
				*auxp2++ = code;
				break;
		    default  : *auxp2++ = *auxp1++;
				break;
		  }
		}
		*auxp2 = '\0';

		*p=insert(*p,value,name);
	}
```

Muestra una función realizada en C que realiza estos tres pasos.

a las mismas desde UNIX en shell, basta referirse a ellas con el signo dólar delante, por ejemplo *$QUERY_STRING*. Si el programa está escrito en C, la forma de acceder es mediante la función *getenv*. En AmigaDOS, se emplea la funcion *GetVar* de la librería DOS y, en general, cada lenguaje (y cada sistema) definirán un método para ello, método que nada tiene que ver con la interfáz CGI ya que todo esto se encuentra en el lado de la aplicación. La interfaz CGI únicamente define la forma de codificación de los datos y el lugar en el que deben estar disponibles: variable de entorno o entrada estándar.

El formato en que se encuentra codificada la información procedente de un form es un flujo de pares:

nombre=valor

separados por el signo "ampersand" (&). Es decir:

nombre_1=valor_1&nombre_2=valor_2 &...&nombre_n=valor_n

En definitiva, se utilice el método que se utilice, habrá que procesar la información recibida según los siguientes pasos:

1. Extraer *tokens* considerando como delimitador el signo "ampersand" (&).
2. En cada token, realizar una nueva separación considerando como delimitador el carácter "igual" (=). Se recuerda al lector que cada campo, botón, o menú en un form se indicaba mediante una etiqueta *<INPUT>* cuyo atributo *NAME* servía para crear una variable que contuviera el valor que le asignaría el usuario al facilitar los datos requeridos [Echeva-1]. El primer token obtenido ahora corresponderá al nombre de la variable (atributo *NAME*) y el segundo, al valor obtenido.
3. *URL-descodificar* (*URLdecode*) los valores obtenidos. La información que recibe el form, ya sea a través de la entrada estándar o de la variable *QUERY_STRING* se encuentra *URL-codificada* (*URLencoded*), lo que significa que los espacios en blanco habrán sido sustituidos por el signo "más" (+) y los caracteres especiales se indican en hexadecimal precedidos por un signo "tanto por ciento": *%xx*. El script CGI deberá, por tanto, recuperar los valores originales.

CONVENIENCIA DE USO DE GET Y POST

Se desaconseja el uso de *GET* para el envío de información de un form a un CGI debido a que se realizará a través de una variable de entorno que puede tener una longitud más o menos limitada según el sistema. Por ello, la información presente en el form podría desbordarla. Imagínese, por ejemplo, el

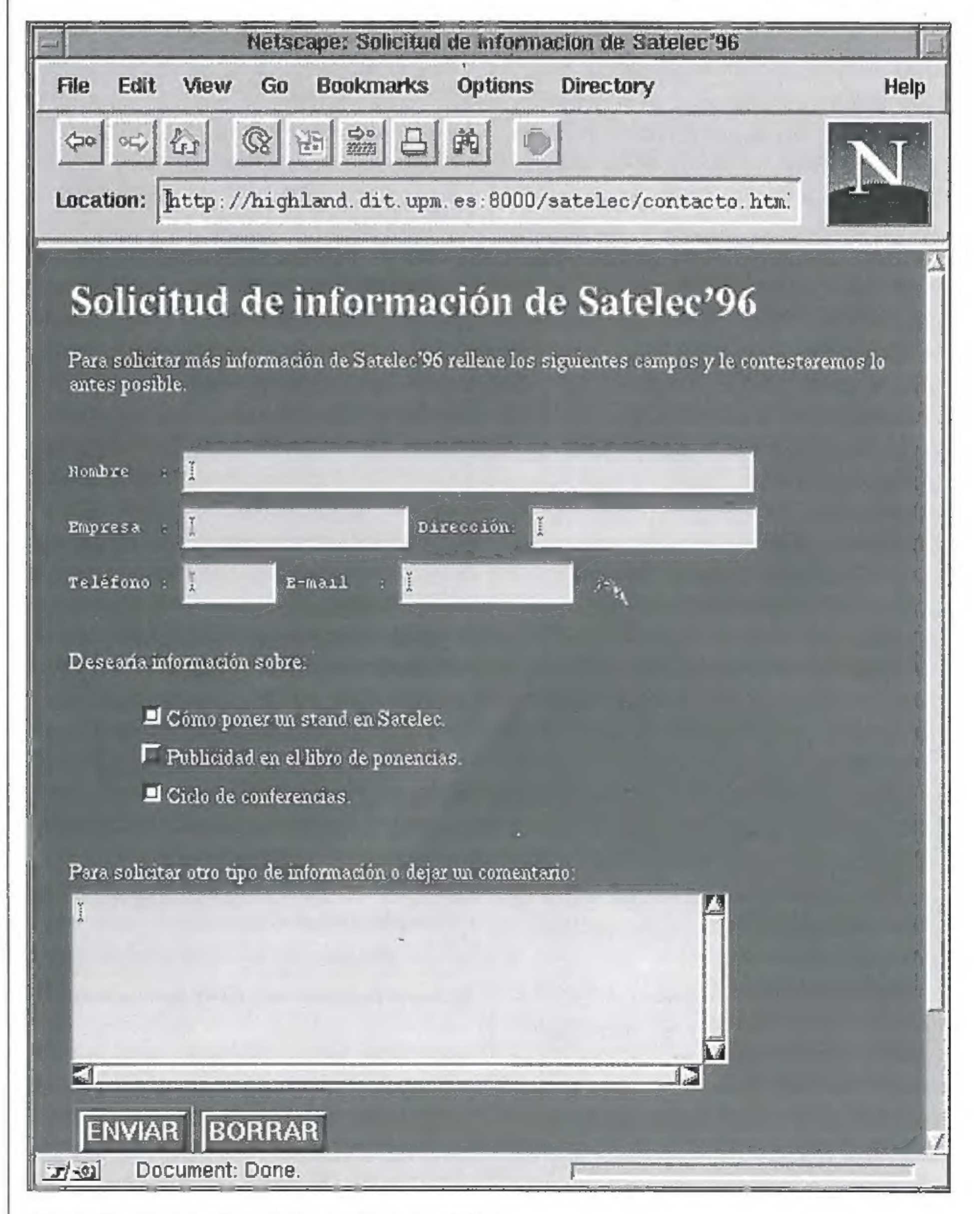


Fig. 1: Ejemplo rudimentario de libro de visitas.

uso de campos *TEXTAREA* (el autor aún se pregunta qué hace este método aquí).

En caso de emplear *POST*, la única aclaración a realizar para aquellos *scripts* que accedan a su entrada mediante una interfáz de ficheros es que el servidor no colocará ningún signo de terminación como, por ejemplo *EOF*. Por ello, si se accede de esta forma a los datos, será necesario emplear la información presente en otra variable de entorno: *CONTENT_LENGTH* que indicará la longitud en bytes de la información que se le envía al CGI. Esta variable se encuentra definida siempre independientemente del método utilizado.

UN CGI-FORM PARA MENSAJES

En el ejemplo del listado 2 se puede observar el código de un CGI realizado en sh encargado de procesar la información del form que se puede observar en la figura 1 y cuya fuente HTML ha sido incluida en el disco de la revista. Este CGI, que sirve para dejar un mensaje en el servidor del autor, no es una pasarela con el correo de Internet, pero construye un fichero */hypertext/messages/mensajes.txt* idéntico a un buzón de correo SMTP, de tal forma que los mensajes se podrán leer con cualquier lector estándar de correo (o simplemente tecleando *mail -f /hypertext/messages/mensajes.txt*). Para crear otro fichero bastará cambiar el valor de la variable *MSG_FICH*. Este form puede ser accedido mediante el URL:
http://highland.dit.upm.es:8000/mensajes.html

Como pasarela con el mail de Internet, el CGI más famoso es uno escrito en *perl* que se incluye en el disco de la revista (para su uso será necesario disponer del perl en el sistema). En cualquier caso, se deja como ejercicio al lector realizar las modificaciones necesarias al del ejemplo para que envíe la información por mail en lugar de almacenarla en un fichero.

Como se puede observar, el CGI lee de la entrada estándar e inicialmente genera las variables necesarias para albergar los datos del form. Mediante las instrucciones *cut* se extraen los campos separados por los delimitadores que se indica en la opción *-d* siguiendo los pasos que se indicaba en los párrafos anteriores. En este sencillo ejemplo, se han ignorado los nombres originales de las variables del form. Después se construye la fecha con el formato específico que se indica en los mensajes de correo SMTP.

LISTADO 2

```
#!/bin/sh

MSG_FICH=/hypertext/messages/mensajes.txt

# Lectura de la entrada estandar (hacia la var.
stdin)
read stdin

# Extraccion de variables
stdin=` echo $stdin | sed "s/+/ /g"`
nombre=` echo $stdin | cut -d"&" -f1 | cut -d"="
-f2`
email=` echo $stdin | cut -d"&" -f2 | cut -d"=" -f2`
subject=`echo $stdin | cut -d"&" -f3 | cut -d"=" -f2`

fecha=`date +%a' '%b' '%d' '%T' '%Y`

echo From $email $fecha     >> $MSG_FICH
echo Return-Path: $email    >> $MSG_FICH
echo From: $nombre          >> $MSG_FICH
echo Subject: $subject      >> $MSG_FICH

# "URL-descodificacion basica"
echo $stdin | cut -d"&" -f4 | cut -d"=" -f2 | sed
"s/%0D//g" \
    | sed "s/%21/!/g" | sed "s/%3F/?/g" \
    | sed "s/%0A/+/g" | tr + '\012' >> $MSG_FICH

echo >> $MSG_FICH

# Inicio de salida de datos: cabeceras
echo Content-type: text/html
echo

cat << FIN
<H1>ENVIO DE MENSAJES</H1>
Hola! Tu mensaje "parece" haber sido enviado,
pero no te fies mucho
por si acaso.<br>
De todas formas:
<B> MUCHAS GRACIAS.</B><P>

Un <B>besito</B> a las <B>nenas</B>
<img src=/icons/rosa.gif align=middle>
  y un <B>apretón de manos</B> a los
<B>nenes</B>.
<p>

<A HREF=/index.html>
<IMG    SRC=/httpd-internal-icons/back.xbm
ALT=Echeva's Server>
</A>

<HR>

 <H1>MESSAGES DELIVERY</H1>
Hi! Your message seems to have been delivered
but, don't trust, maybe... <br>
Anyway:
<B> THANK YOU VERY MUCH.</B><P>

<B>Little Kisses</B> to all the <B>girls</B>
<img src=/icons/rosa.gif align=middle>
& a strong <B>handshake</B> for <B>boys</B>.
<p>

<A HREF=/index-e.html>
<IMG    SRC=/httpd-internal-icons/back.xbm
ALT=Echeva's Server>
</A>
```

Ejemplo rudimentario de libro de visitas.

LISTADO 3

```
/* parser.c - Example use of functions CGIparser,
list_process */

#include <stdio.h>
#include "CGIparse.h"

#define MAXLEN 1024

int process(char *name, char *value);

main()
{
        char input[MAXLEN];
        struct pair *root = NULL;

        scanf("%s",input);

        CGIparser(input,&root);
        list_process(root,process);
}

process(char *name, char *value)
{
        printf("%s=%s\n",name,value);
}
```

Ejemplo de uso de Echeva's CGIparse.

Tras ello, se realiza una pequeña parte de la tediosa tarea de *URL-descodificar* los datos, sustituyendo la codificación hexadecimal de algunos caracteres por los caracteres ASCII correspondientes. Con el ejemplo, sólo le leerán correctamente los mensajes en inglés y sin signos de entonación, ya que no se han previsto casos como acentos, aperturas de exclamación e interrogación, etc.

Por último, el CGI devuelve "algo" por la salida estándar. Recuérdese que esto es un paso obligatorio. En este caso, se trata de una página HTML de confirmación, aunque si se desea no mostrar ninguna página, siempre es posible emplear una cabecera *Location* (ver artículo del mes anterior).

PROGRAMAS DE AYUDA

Como se ha podido comprobar por el anterior ejemplo, la parte más tediosa del procesamiento de *forms* corresponde a la fase de URL-descodificación de los datos. Como casi siempre que uno quiere hacer algo, ya lo ha hecho antes otro y muchas veces mejor. Además, si se trata de un colega de Internet, es bastante probable que sea gratis. En efecto, se puede encontrar código que realiza estas funciones para:
Bourne Shell: The AA archie gateway, CERN cgiparse
C: *Scripts* de NCSA httpd, Echeva's

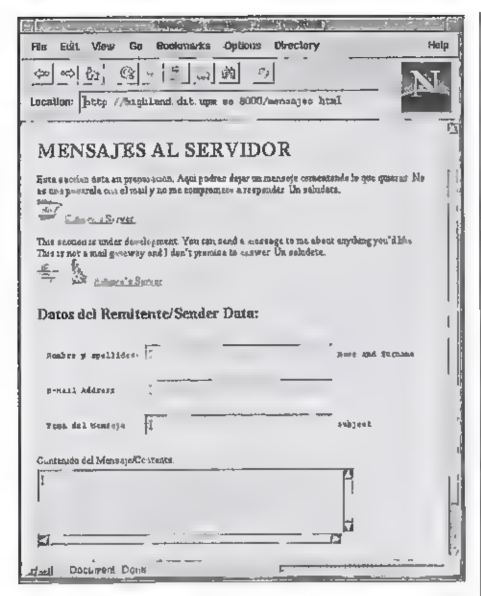


Fig. 2: Form para recogida de información.

CGIparse
PERL: The PERL CGI-lib
PERL5: CGI.pm
TCL: TCL argument processor

ECHEVA'S CGIPARSE

Uno de los colegas de Internet que se ha encontrado ya con el problema de la URL-descodificación es el autor de este artículo, por lo que en el disco de la revista se ha incluido un módulo denominado *sp-parse.tgz* del que proviene el código del listado 3.

En él, se definen dos funciones:

CGIparser(cadena,lista_var)

que toma una cadena URL-codificada y devuelve una lista de pares (nombre, valor) para cada variable de form que se pueda extraer de la cadena proporcionada. En la función, *cadena* es la dirección de la cadena a URL-descodificar y *lista_tokens* es un puntero a una lista de nodos *struct pair* que no son otra cosa que una estructura con dos campos: uno para el nombre y el otro para el valor de cada variable.

list_process(lista_var, función_recorrido)

que recorre la lista que se le indica en *lista_var* y en cada nodo aplica la función *función_recorrido*, programada por el usuario, que debe aceptar dos parámetros que sean direcciones donde depositar los valores de los campos de *struct pair*.

En el ejemplo del listado 4 se puede apreciar un simple programa que toma una cadena URL-codificada de la entrada estándar y la URL-descodifica haciendo uso de las funciones de CGIparser. Para comprender mejor su uso pruebe a ejecutarlo para analizar la cadena:

var1=HOLA+AMIGO%21&var2=%7Eecheva&var3=5%2B2

LA UTILIDAD CGIPARSE

El servidor de WWW que se entregó con el número 19 de la revista incluye, además, la utilidad *cgiparse* que puede ser utilizada desde cualquier shell de UNIX. Si no ha instalado este servidor y utiliza el de otra persona o el de su organización, este programa suele encontrarse en el directorio de binarios o en el de CGIs del servidor.

La sintaxis de esta utilidad responde a la forma:

cgiparse [lista de [-opción [modificador] ...]

donde, entre las principales opciones cabe destacar:

keywords

Considera *QUERY_STRING* como una lista de parámetros. Los URL-descodifica y los presenta en la salida estándar, uno por línea.

form

Considera *QUERY_STRING* como datos provenientes de un form. Devuelve una cadena que si se evalua desde una Bourne shell mediante el comando *eval* (ver listado 4), genera variables de shell con los valores que el usuario ha proporcionado en el form. Estas variables se denominarán *FORM_nombre*, donde *nombre* es el nombre de la variable del form (correspondiente al campo *NAME*).

read

Simplente lee *CONTENT_LENGTH* caracteres de la entrada estándar y los devuelve por la salida estándar.

init

Si la variable *QUERY_STRING* no se encuentra definida, lee la entrada estándar y devuelve una cadena que si se evalúa desde una Bourne shell mediante el comando *eval* (ver listado 4), genera una variable *QUERY_STRING* con los valores adecuados. Es de gran utilidad para CGIs que procesen *forms* que puedan utilizar cualquiera de los dos métodos *GET* o *POST*. Para ello, basta con que el script genere mediante esta opción la variable si no existe (método *POST*) y luego proceda de igual modo en ambos casos. Esto, además, permite que se llame varias veces a *cgiparse* desde un mismo script. Si no fuera así, y el método empleado fuera *POST*, únicamente se podría llamar a *cgiparse* una vez, en la que leería toda la información de la entrada estándar y la siguiente llamada se bloquearía a la espera de más información.

LISTADO 4

```
#!/bin/sh

CGI_DIR=/home/www/cern/server/scripts
MSG_FICH=/hypertext/messages/evap.txt

if [ "$REQUEST_METHOD" = POST ] ; then
    eval `$CGI_DIR/cgiparse -init`
fi

eval `$CGI_DIR/cgiparse -form`

fecha=`date +%a' '%b' '%d' '%T' '%Y`

echo From $FORM_email $fecha >> $MSG_FICH
echo Return-Path: $FORM_email >> $MSG_FICH
echo From: $FORM_nombre >> $MSG_FICH
echo Subject: SATELEC >> $MSG_FICH

echo >> $MSG_FICH

echo "Nombre : $FORM_nombre" >> $MSG_FICH
echo "Empresa : $FORM_empresa" >> $MSG_FICH
echo "Direccion: $FORM_direccion" >> $MSG_FICH
echo "Telefono : $FORM_telefono" >> $MSG_FICH

echo >> $MSG_FICH

echo Solicita informacion sobre: >> $MSG_FICH
echo "$FORM_stand $FORM_public $FORM_conferen" >> $MSG_FICH

echo >> $MSG_FICH

echo $FORM_comentario | tr '' '\n' >> $MSG_FICH

echo >> $MSG_FICH

echo Content-type: text/html
echo

cat << FIN
<H1>ENVIO DE MENSAJES</H1>
La petici&oacute;n ha sido enviada. En breve recibir&aacute; respuesta.
<br>

<B> MUCHAS GRACIAS </B>

<A HREF=/index.html> Para volver a la p&aacute;gina principal
</A>

<HR>
```

CGI para recogida de información empleando cgiparse.

value <nombre_campo>

Considera *QUERY_STRING* como resultado de un form y devuelve únicamente el valor del campo de nombre *nombre_campo*.

La utilidad *cgiparse* cuenta además, con una serie de modificadores de menor utilidad, por lo que no se comentarán en este artículo.

Como valores de retorno la utilidad devuelve:

0: Éxito

1: Línea de comando no válida

2: Variables de entorno con valores incorrectos

3: No se ha podido recuperar la información deseada. Por ejemplo, no existía el campo solicitado, *QUERY-STRING* contiene parámetros en lugar de información procedente de un form, etc.

El principal inconveniente de esta utilidad es el uso que desde una shell requiere de *eval*, lo que puede plantear un problema de seguridad. Si se emplea desde un programa C, al tratarse de una utilidad compilada y no de un módulo de biblioteca, requiere llamadas al sistema para la ejecución de la misma, conexión con su descriptor de salida y análisis de ésta, por lo que, en realidad, no simplifica el programa.

El listado 4 muestra un script mejorado realizado por el autor y Eva M. Castro para envío de mensajes al que se puede acceder en *http://highland.dit.upm.es:8000/stelec/info.html* que hace uso de la utilidad *cgiparse*. El aspecto del form puede observarse en la figura 2 y su fuente HTML en el disco de la revista. En él, primeramente se comprueba el valor de otra variable de entorno definida por el estándar denominada

REQUEST_METHOD

y que proporciona el método empleado. El script funcionará tanto si se hace uso del método *GET* como si se emplea *POST*. En este último caso, genera "artificialmente" la variable de entorno *QUERY_STRING* mediante la opción *init* de *cgiparse*. Una vez exista esta variable, el proceder será idéntico sea cual sea el método utilizado: se empleará la opción *form* para recuperar los valores de las variables del form. En este listado se ha definido una variable *CGI_DIR* que el lector debera modificar para indicar el directorio de su sistema en que se encuentra la utilidad *cgiparse*.

LISTADO 5

```
#!/bin/sh

CGI_DIR="/home/www/cern/server/scripts"

if [ "$REQUEST_METHOD" = POST ] ; then
    eval `$CGI_DIR/cgiparse -init`
fi

eval `$CGI_DIR/cgiparse -form`

res=`expr "$FORM_op1" "$FORM_op" "$FORM_op2"`

echo "Content-Type: text/html"
echo ""
echo "<HTML>"
echo "<H1>Calculadora b&aacute;sica</H1>"
echo "<B>Resultado:</B>"
echo "$FORM_op1 $FORM_op $FORM_op2 = $res<p>"
echo "<A HREF=/form.html>CONTINUAR</A>"
echo "</HTML>"
```

Calculadora CGI en Bourne Shell.

LOS FORMS COMO MECANISMO DE ENTRADA

Aunque la mayoría de las veces que se emplea un form es para recoger cierta información y almacenarla en una base de datos, hay que tener en cuenta que se trata del único método de entrada que se permite al usuario. Por ello, serán siempre necesarios en todo programa interactivo. Como ejemplo, se ha desarrollado una simple calculadora con las cuatro reglas básicas que opera únicamente con enteros. El fuente HTML de la página se ha incluido en el disco de la revista. En este simple programa, se emplea un form con campos de texto para la recogida de los operandos y botones mutuamente excluyentes (*radio buttons*) para selección del operador.

Con el fin de mostrar una vez más la independencia del lenguaje respecto a la interfaz CGI, se ha desarrollado el ejemplo en dos lenguajes. El listado 5, escrito en Bourne shell, hace uso de la utilidad *cgiparse* y está preparado para procesar el form independientemente de si se emplea un método *GET* o *POST*, como se hizo con el ejemplo de mensajes.

El listado 6, escrito en C, hace uso del módulo de biblioteca *CGIparse* del autor. En el disco de la revista se facilita un Makefile junto a la fuente *formcgi.c*. Para compilarlo será necesario extraer el contenido de *sp-parse.tgz* en el directorio en que se encuentren estos dos ficheros.

LISTADO 6

```
/* formcgi.c : calculadora CGI (echeva@dit.upm.es) */

#include <stdio.h>
#include <string h>

#include "CGIparse.h"

int args_index=0;
char *args[3],

process(char *name, char *value)
{
      args[args_index++] = strdup(value);
}

main(int *argc, char **argv)
{
    int op1, op2, res;
    char *op;
    char input[256];
    struct pair *root = NULL;


fgets(input,atoi(getenv("CONTENT_LENGTH"))+1,stdin);
    CGIparser(input,&root);
    list_process(root,process);

    op1=atoi(args[0]);
    op=args[1];
    op2=atoi(args[2]);

    switch (*op) {
            case '+' : res=op1+op2; break;
            case '-' : res=op1-op2; break;
            case '*' : res=op1*op2; break;
            case '/' : res=op1/op2; break;
    }

    printf("Content-Type: text/html\n\n");
    printf("<HTML>\n");
    printf("<H1>Calculadora b&aacute;sica</H1>\n");
    printf("<B>Resultado: </B>");
    printf("%d %c %d = %d<p>\n",op1,*op,op2,res);
    printf("<A HREF=/form.html>CONTINUAR</A>\n");
    printf("</HTML>\n");
}
```

Calculadora CGI en C.

En este caso, el programa únicamente servirá para procesar un form que utilice el método *POST* ya que lee de la entrada estándar. Se deja al lector como ejercicio que lo modifique para procesar un form con método *GET*. En este caso, se ha utilizado además, como ejemplo, una lectura del flujo de entrada estándar y, como se indicaba anteriormente, ha sido necesario tener en cuenta el valor de la variable de entorno *CONTENT_LENGTH* para leer únicamente los datos que proporciona el servidor. Este detalle es importante ya que el scipt podría encontrar en el flujo de entrada más datos de los que realmente le son útiles, y que tendrían un valor indeterminado por lo que su procesamiento podría

dar lugar a errores aleatorios.

Para utilizar uno u otro bastara renombrarlos como *formcgi* y colocarlos en el directorio de ejecutables del servidor.

CAMPOS OCULTOS

Dado que el HTTP es un protocolo de transmisión sin estado, una vez "pintada" la página por el cliente se habrá perdido toda clase de conexión con el servidor. Así pues, éste no será consciente de ningún tipo de sesión y se encontrará completamente desligado del cliente.

Imagínese que a lo largo de una sesión con un servidor de WWW es necesario mantener un pequeño estado de lo que se está haciendo. Este pequeño estado puede ser algo tan simple como conocer cuál fue el último movimiento de quién realiza ahora una conexión para actuar en consecuencia. Por poner un ejemplo, supóngase que en el anterior ejemplo de la calculadora se deseara que cada vez que se facilita el resultado vuelva a aparecer el form. Bastaría con que el CGI que procesa el form mostrara el código HTML correspondiente por la salida estándar junto al resultado.

Imagínese ahora que se desea que en la nueva calculadora aparezcan en los campos de datos los operandos que se teclearon la vez anterior. No hay problema: bastará que el CGI introduzca un atributo *VALUE* con los valores en las etiquetas *<INPUT>* correspondientes (si no le queda claro repase [Echeva-1]). En este caso, una vez se muestre la página ya no habrá conexión, pero aún se dispondrá de información del anterior paso en el form y, si el usuario no la modifica, será de nuevo enviada al pulsar el botón correspondiente. Es decir, se habrá transmitido información de estado (se proponen estos cambios como ejercicio).

Problema: "el usuario ve la información y, lo que es peor, la puede modificar".

Solución: "hacer que no la vea y no la pueda modificar".

Así es como se introdujo el tipo *HIDDEN* para el atributo *TYPE* de la etiqueta *<INPUT>*. De esta forma, cualquier información de estado que se desee transmitir puede ser incluida en un form de forma que no aparezca visible ni modificable al usuario y, cuando éste active el form, será retransmitida de nuevo al servidor.

Así, por ejemplo, en el WebTalk que está desarrollando el autor, aún en fase alfa en su servidor, al entrar al foro el usuario debe identificarse mediante un form similar al siguiente:

```
<FORM METHOD=POST
 ACTION=/cgi-bin/ejemplo>
Escribe aquí tu nombre:
<INPUT TYPE=TEXT NAME=name>
</FORM>
```

Posteriormente, los programas CGI de comunicaciones multiusuario hacen uso de este nombre para indicar en pantalla quién habla y devuelven un form para que el usuario introduzca el texto en el que el nombre se transmite mediante un campo *HIDDEN*:

```
(...)
printf("Content-type: text/html\n\n");
printf("<FORM METHOD=POST
 ACTION=%s>\n",CGI_SEND_NAME);
printf("<INPUT TYPE=HIDDEN
 NAME=name VALUE=%s>\n",args[1]);
printf("<INPUT TYPE=TEXT NAME=
sentence SIZE=80>\n");
printf("</FORM>\n");
(...)
```

BIBLIOGRAFÍA

[Echeva-1] Fernando J. Echevarrieta. "El lenguaje HTML (II)", Sólo Programadores n.17 Pág. 10.

CONCLUSIÓN

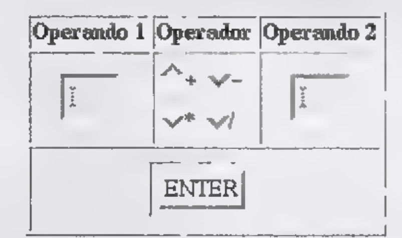
Con el aspecto de los campos ocultos se da por finalizado el tema de *forms*. Sin embargo, se comienza a vislumbrar cómo será necesario emplear ciertos trucos para programar con CGI. De la interfáz queda poco más por ver: el paso de parámetros a través de URLs y la realización de documentos ISINDEX. En próximos artículos se presentarán estos últimos detalles y se realizara una formalización de la interfáz basada en el draft del 8 de Enero de 1996 (o de una futurible revisión). Simultáneamente, se tratará que el lector domine el paradigma de programación con CGI mediante el desarrollo de aplicaciones ejemplo que muestren los distintos trucos que proporciona la experiencia y no están escritos.

PROXIMAMENTE

El abanico de tecnologías WWW se amplia cada vez más, los estándares (de facto) se suceden y algunos toman especial fuerza como Java, especialmente cuando surgen asociaciones entre las firmas comerciales más poderosas, como es el caso de SUN, Netscape Corp. y Silicon Graphics. Así pues, es necesario penetrar en todas estas tecnologías simultáneamente. Aún queda mucho que decir sobre CGI; sin embargo, con el objeto de no dejar a un lado ninguna tecnología, el próximo artículo será una introducción a Java. A partir del mes próximo, la linealidad de esta serie se flexibilizará: los artículos sobre programación CGI y Java se irán alternando, abriendo algunos paréntesis para cubrir los nuevos avances en HTML o en otras tecnologías, aún no comentadas, como VRML.

Calculadora básica

Este es un ejemplo de empleo de forms para entrada de datos a un script CGI. De esta forma se logra un interfaz de entrada. Como ejemplo se ha tomado una calculadora con las cuatro reglas básicas que sólo admite enteros. Si no se opera con enteros da error.



Autor: Fernando J. Echevarrieta

Figura 2. Ejemplo de Form para entrada de datos.



EL PROCESO DE ANÁLISIS Y DISEÑO

José C. Remiro

Noralmente, el programador forma parte de un grupo de trabajo que tiene como finalidad construir el sistema de información de la empresa para la que trabaja. Este artículo pretende mostrar las distintas fases por las que pasa un proyecto para construir un sistema, hasta que se obtienen las especificaciones de los programas.

Este artículo intentará explicar cómo se puede construir un sistema, indicando algunos métodos utilizados para el análisis y el diseño, de tal forma que, al final de éste, se den diversas especificaciones para proceder a su implementación.

Se entenderá por sistema "un conjunto de recursos y métodos que se encuentran organizados para realizar determinadas funciones". Así, dentro de una empresa se podría encontrar un sistema de control de inventario, encargado de asegurar en todo momento que la empresa dispone de los elementos necesarios para realizar sus funciones. El conjunto de recursos de este sistema comprendería, entre otros, a los empleados que trabajan en tales funciones, los diversos medios de almacenamiento de información y los elementos para tratarla (archivos, medios magnéticos, formularios, etc.). El conjunto de procedimientos que se aplican a la información y las normas de trabajo corresponderían a los métodos del sistema.

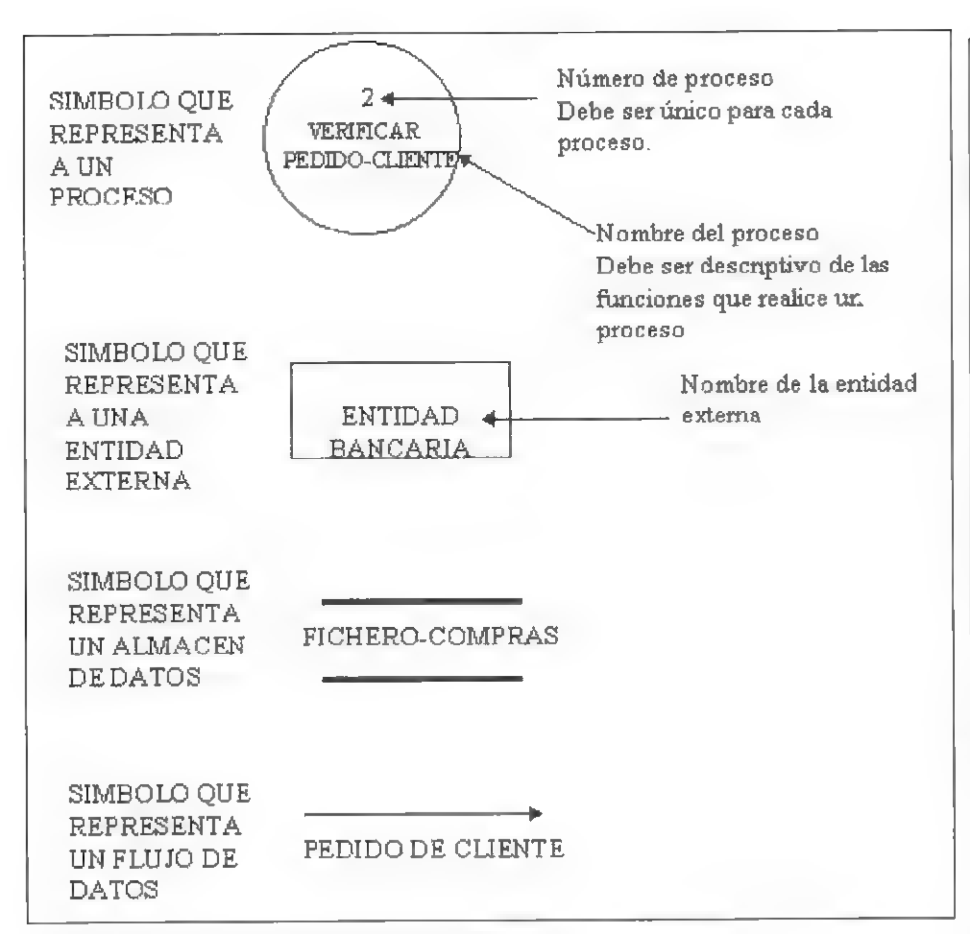
Los pasos sucesivos que normalmente se siguen para la construcción de un sistema son:

Análisis: Dependerá de si el sistema existe o no. Si el sistema ya existe, sirve para conocer qué hace y qué necesidades debe satisfacer. Si, por el contrario, el sistema no existe, sirve para obtener información sobre el conjunto de descripciones y requerimientos que debe cumplir.

Diseño: Tomando como punto de partida el análisis realizado anteriormente, el diseño intentará proponer la construcción del nuevo sistema. Para ello, puede partirse del sistema ya existente o bien realizar la construcción completa de uno nuevo.

Implementación: Una vez concluido y aprobado el nuevo sistema, esta es la fase en la que se realiza la contrucción del mismo.

En la creación y desarrollo de los sistemas informáticos no sólo participa el personal informático. Puesto que será necesario saber cómo funciona el sistema, durante las etapas de análisis y diseño habrá que tener especialmente en cuenta a los usuarios, siendo éstos los interesados en la información que producirá el sistema. Por tanto, son muy importantes para ellos los criterios de corrección (el usuario obtiene los datos de forma correcta y con un tiempo de respuesta adecuado) y fácil interacción con el sistema. La dirección de la empresa a la que pertenece el sistema es otra parte que interviene en el desarrollo de éste, estando interesada en que sea eficiente, eficaz y, además, que el proyecto se ajuste a un presupuesto y a un calendario adecuados. Por último, los integrantes del centro de proceso de datos de la empresa se encargarán del desarrollo del sistema. Con los avances tanto de la informática personal como de las comunicaciones, el centro de proceso de datos se ha ido descentralizando, pasando a su vez a estar todo su personal distribuido por los diversos departamentos que integran la empresa. Incluso, también hay usuarios (denominados usuarios finales) que participan en el desarrollo de parte del sistema, normalmente dentro del departamento al que pertenecen, siendo el centro de proceso de datos el encargado de desarrollar la parte del sistema que utilizan todos los departamentos de forma conjunta.



Cuadro1: representación de los símbolos utilizados.

EL CICLO DE VIDA

Para poder organizar todas las actividades relacionadas con la construcción de un sistema y especificar cada uno de los pasos de este proceso, se suele utilizar el ciclo de vida del sistema. La finalidad de éste, es ayudar a las personas involucradas en el proyecto a resolver los diferentes problemas que van surgiendo, así como el de supervisar los resultados. Además, el ciclo de vida resulta de gran ayuda si se tienen que realizar informes dirigidos a la dirección y mantener un control de los recursos utilizados. En la práctica, suelen utilizarse diversos ciclos de vida. A continuación se comentará uno de los más utilizados: *el ciclo de vida lineal*. Este ciclo de vida se compone de una serie de fases consecutivas. Cada fase no puede empezar hasta que la fase previa haya sido terminada. Además, cada fase terminará con un informe que será el punto de partida de la siguiente fase. Dicho informe incluirá descripciones del sistema, decisiones de diseño, así como los problemas que se hayan encontrado. Las fases de que consta el ciclo de vida lineal son: Definición del problema, estudio de viabilidad, análisis, diseño y construcción del sistema.

Durante la fase de definición del problema se especifican los problemas a resolver, se determinan los límites del proyecto y se decide la composición del grupo que dirigirá el proyecto, así como los recursos que le serán asignados al mismo para la construcción del sistema.

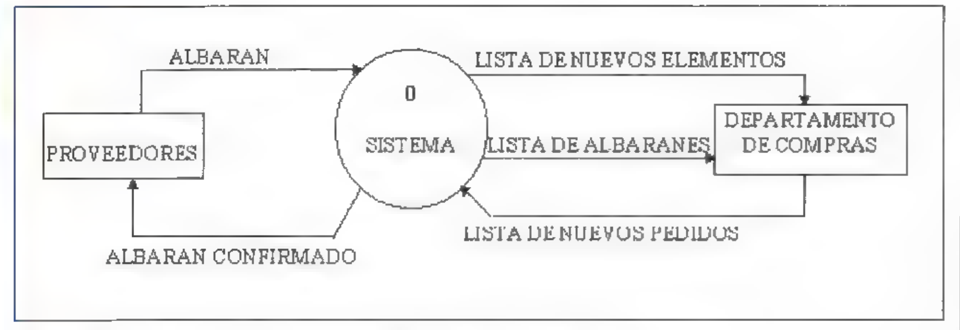
El documento de salida de esta fase se compondrá de un presupuesto, el personal que intervendrá en cada una de las fases, las fechas probables de inicio y de finalización del proyecto, así como las partes del sistema que serán modificadas y aquellas que no variarán, quedando tal y como están.

El ciclo de vida lineal se compone de una serie de fases consecutivas que no comienzan hasta que la anterior no haya concluido

Durante el estudio de viabilidad (segunda fase), es cuando se proponen varias soluciones al problema. Evidentemente, estas soluciones son muy generales, pues se está al comienzo de la construcción del sistema y aún no se sabe si se llevará a cabo o no su construcción.

Una vez propuestas las diferentes soluciones, se estudia su viabilidad respecto a tres criterios diferentes. Se debe comprobar que económicamente es viable, es decir, si los recursos invertidos en el proyecto se recuperarán en un futuro determinado. Si es técnicamente factible, es decir, si la empresa tiene los recursos y conocimientos suficientes para realizar el proyecto. Por último, su viabilidad operacional, que consiste en saber el impacto que tendrán las modificaciones del sistema o la creación del nuevo sistema. La salida de esta fase la constituye un documento especificando, en líneas generales, la solución elegida, caso de no haberse desechado el proyecto, los beneficios y los costos esperados, así como una completa y detallada lista de todos los recursos que van a ser empleados en cada una de las fases.

La fase de análisis realiza un estudio detallado del sistema para que el equipo de diseño estudie su funcionamiento. Para ello, los analistas obtienen información de diversas fuentes (entrevistas, cuestionarios, revisión de documentos que se generan, etc.) y estudian los componentes del sistema (bases de datos, cómo opera el sistema existente, etc.). Como salida de esta fase se obtiene un modelo, que constará de las funciones que realiza, de los datos que utiliza y de los flujos de información. Además, como hay un mayor conocimiento del sistema, es posible conocer los problemas con más detalle (éstos se identificaron en la primera fase), por lo que pueden surgir nuevos objetivos y



Cuadro 2 diagrama de contexto para un sistema de pedidos.

una solución para alcanzarlos. Además, se establece una nueva estimación de los costes del proyecto, en esta ocasión, mucho más aproximados.

Durante la fase de diseño, los diseñadores deben seleccionar el equipo informático que se utilizará, especificar los nuevos programas y las modificaciones que se realizarán sobre los anteriores. También se diseñarán las nuevas bases de datos, así como las posibles modificaciones sobre las ya existentes. Además, en esta fase también se describirá la forma en que los usuarios utilizarán el nuevo sistema (denominados procedimientos de usuario). Puesto que la fase de diseño es bastante complicada, puede dividirse en dos pasos más sencillos, si el proyecto es complejo: el diseño estructurado y el diseño detallado. Durante el diseño estructurado se identifican el conjunto de flujos de datos que interactuan con el exterior, se proponen las nuevas funciones del sistema y las modificaciones a realizar sobre las ya existentes.

Para construir un sistema, previamente hay que conocer las necesidades que van a tener de los futuros usuarios

Por último, se detallarán las partes que se automatizarán y las que se informatizarán. Una vez tomada esta última decisión, es cuando puede comenzar el diseño detallado. En este paso, se diseñan los módulos de los programas, las bases de datos y se construyen los manuales de operación para el usuario, definiendo las interfases entre los ordenadores y los usuarios, de tal forma que se defina qué debe hacer el usuario para poder utilizar el sistema.

La salida de la fase de diseño comprende una configuración del equipo informático, especificaciones de la base de datos y de los módulos de los programas. Además, se incluyen los manuales de operaciones para los usuarios, los formularios y las interfases entre el usuario y el ordenador totalmente detallados.

En la creación y desarrollo de los sistemas informáticos participan tanto el personal informático como los futuros usuarios

La fase de construcción del sistema puede dividirse a su vez en dos fases más pequeñas: el desarrollo y la implementación.

Durante el desarrollo se escriben y se prueban los programas, comprobándose éstos de forma individual por parte de los usuarios. Además, se inicializan las bases de datos. Durante la implementación se pone en uso el nuevo sistema. Previamente, los usuarios deben haber sido entrenados en las diferentes partes que hayan sido modificadas.

Al final de esta fase, los usuarios ya tienen un nuevo sistema, incluyendo las bases de datos correctas y los programas de trabajo. Además, se debe entregar la documentación sobre cada uno de los programas que se han sido desarrollados.

Tras la puesta en marcha del sistema hay que realizar dos actividades muy importantes: el mantenimiento y la post-implementación. El mantenimiento trata de eliminar errores del sistema durante su existencia. Además, es posible que se produzcan variaciones en el entorno de éste. Si son necesarios cambios que afecten a una parte considerable del sistema es posible que halla que realizar un nuevo proyecto revisando todas las partes del ciclo de vida. La post-implementación consiste en la evaluación del nuevo sistema para comprobar si los objetivos se han alcanzado y si se han conseguido los beneficios esperados. La post-implementación, a menudo, puede dar lugar a modificaciones en el sistema. Además, se pueden revisar decisiones tomadas durante el proyecto para poder mejorar la realización de proyectos futuros.

De todas formas, el ciclo de vida lineal para el desarrollo de sistemas presenta algunos inconvenientes. Cada fase no puede realizarse hasta que las anteriores no hayan terminado, puesto que el tipo de desarrollo es top-down (de lo general a lo particular). En fases avanzadas del desarrollo se pueden encontrar errores respecto al plan original del proyecto y, por tanto, es necesario revisar las fases anteriores. Una de las causas que pueden llevar a la anterior situación es que el problema a resolver es demasiado complejo o poco concreto por lo que puede ser preferible dividirlo en subproblemas más sencillos.

INICIO DE UN PROYECTO

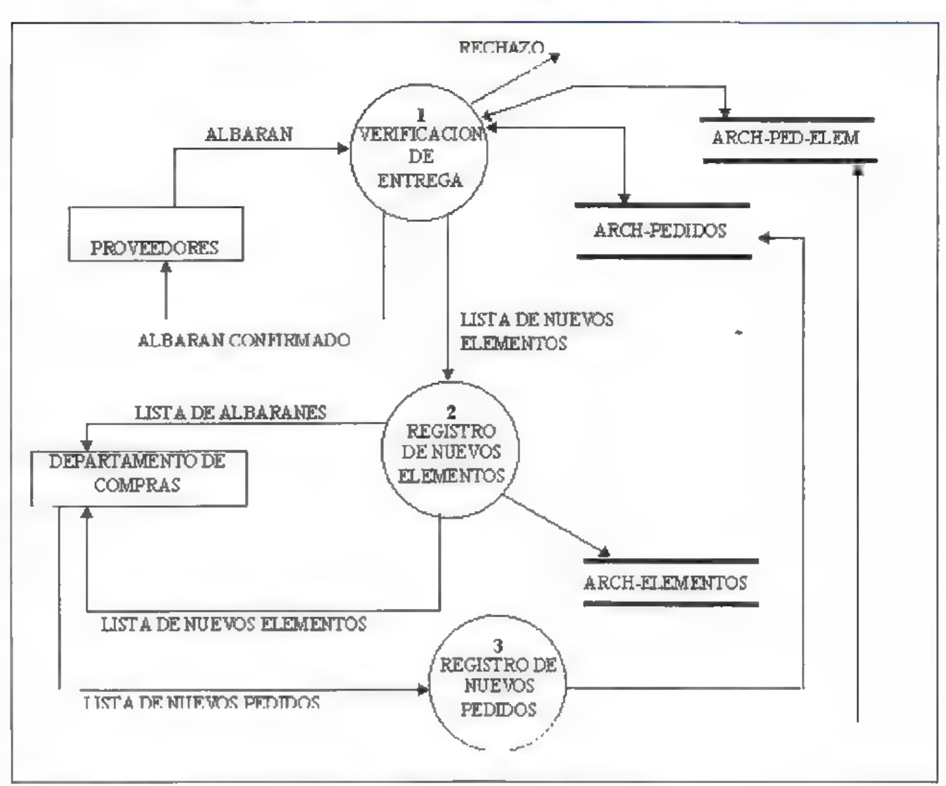
No es posible la construcción de un sistema sin conocer las necesidades que tienen los usuarios y qué servicios



esperan conseguir. Por tanto, se necesita la recogida de información. Para que ésta sea completa y no redundante, será necesario identificar las fuentes y las técnicas de recogida, indicando para cada fuente qué técnica va a ser utilizada.

Las fuentes de información más usuales suelen ser los usuarios, los formularios, los documentos, los programas, los manuales de trabajo y los informes. De esta forma, consultando a los usuarios, se podrán determinar las diversas actividades que llevan a cabo, así como, los servicios que esperan obtener del sistema. De los formularios y documentos se pueden obtener las transacciones (entradas al sistema que provocan modificaciones en las bases de datos) y los flujos de datos, así como los datos elementales y las relaciones entre ellos. Los programas permitirán ver qué funciones realizan y qué estructuras de datos manejan. Los manuales de trabajo (si los hay), especifican qué labores realizan cada uno de los empleados de una organización. De esta forma, pueden utilizarse para conocer en detalle qué hace cada uno de los usuarios. De los informes pueden obtenerse las diferentes salidas del sistema. Cuando hay que construir un nuevo sistema, es evidente que se disponen de pocas fuentes de información. Casi la única serán los usuarios, que indicarán qué es lo que esperan del nuevo sistema. Es buena idea estudiar sistemas que realicen funciones similares.

En la fase de definición del problema, a partir de la información recopilada, se deben establecer una serie de objetivos. Normalmente, estos objetivos están relacionados con diferentes problemas del sistema que actualmente está operando. De esta forma, pueden ser necesarias nuevas funciones, bien pueden existir funciones que realice el sistema pero éstas no se realizan de forma satisfactoria o bien tienen un coste muy alto. Es evidente que los objetivos seleccionados se pueden solucionar de diferentes formas. Cada una de ellas se evaluará durante el análisis de viabilidad, aplicando durante esta fase un análisis sobre el coste de las diferentes soluciones. Por último, se selecciona la solución que se considera más adecuada.



Cuadro 3 dfd de alto nivel para el diagrama del cuadro 2

Cada fase terminará con un informe, el cual servirá como punto de partida de la siguiente fase

ANÁLISIS

Una vez que se ha seleccionado una solución junto con los correspondientes objetivos, el siguiente paso consistirá en realizar un modelo sobre el sistema existente. Previamente puede haber sido necesario volver a recopilar más información.

Una de las herramientas más difundidas para la modelización son los diagramas de flujos de datos. Están formados por una serie de símbolos que tienen como función representar cada uno de los elementos que forman parte del sistema. Básicamente existen símbolos para representar procesos (describen las funciones que realiza el sistema), los almacenes de datos (cualquier elemento capaz de almacenar datos), entidades externas (entidades que interactuan con el sistema) y, por último, los flujos de datos (describen los movimientos de información que se producen en el sistema). Como el lector podrá observar, en el cuadro 1 están representados cada uno de los símbolos utilizados.

Para modelizar el sistema se comienza construyendo el diagrama de contexto, que representa los movimientos de información entre el sistema y las entidades externas. Así, solamente se dispondrá en este diagrama de un único proceso que representará a todo el sistema a analizar, junto con los flujos de datos que se dirigen o provienen de las entidades externas. Estas entidades externas pueden pertenecer a la empresa o no. Si el sistema a analizar forma parte de otro más grande, las partes de éste que interaccionan con el subsistema a analizar pueden ser consideradas como enti-

dades externas. En el cuadro 2 se muestra un diagrama de contexto para un sistema de recepción y almacenado de pedidos.

Una vez construido el diagrama de contexto, se procede a detallar cómo trabaja el sistema. Así, el proceso que representa a éste es sustituido por un conjunto de subprocesos que expresan tareas más específicas y que, conjuntamente, representan al sistema, pero con más detalle (diagrama de flujo de datos o dfd de alto nivel). Junto a éstos, aparecerán flujos de información entre los diversos subprocesos, además de los almacenes de datos que utilizan. Es muy importante que el diagrama de flujo de datos sea coherente con el diagrama de contexto. Esto quiere decir, entre otras cosas, que los flujos de datos y las entidades externas que aparecieron en el diagrama de contexto volverán a aparecer en el diagrama de flujo de datos de alto nivel. En el cuadro 3, aparece un diagrama de flujo de datos de alto nivel para el diagrama de contexto del cuadro 2.

El diseño de un nuevo sistema tiene menos inconvenientes que el "rediseño" de un sistema ya existente

El procedimiento anterior puede realizarse tantas veces como sea necesario para cada proceso del diagrama de flujo de datos de alto nivel, de tal forma que, al final se obtenga un modelo que explique de forma esquemática cómo funciona el sistema. Sólo habrá que considerar que los diferentes diagramas deben ser coherentes y que no merece la pena que los procesos se detallen totalmente.

También hay que tener en cuenta que, a medida que se desarrolla el modelo, es conveniente realizar un análisis de datos, que estará representado en el diccionario de datos, consistente en una descripción de las características (longitud, tipo, información lógica que albergan, etc.), estructura (descripción de flujos de datos y registros lógicos) y relaciones existentes entre los datos (utilizando, por ejemplo, el modelo entidad-relación).

DISEÑO

Esta fase comienza basándose en el modelo del sistema producido anteriormente, es decir, durante la etapa de análisis. La fase de diseño tiene como fin la construcción de un sistema a partir del existente, o bien, de un nuevo sistema, si es que éste no existe. Como es evidente, este nuevo sistema debe cumplir todos y cada uno de los objetivos que han sido fijados en las fases anteriores.

El primer paso a dar durante esta fase será identificar las diferencias entre el modo de trabajo y las funciones del sistema actual, así como los nuevos modos de trabajo y las nuevas funciones que se desean para el sistema a construir. De esta forma, se selecciona en primer lugar el equipo informático, y se decide qué operaciones se realizarán manualmente y cuales se automatizarán seleccionando los lenguajes de programación, los sistemas gestores de bases de datos y los paquetes informáticos que se utilizarán para la implementación. Posteriormente, se procede a identificar el dominio del cambio, compuesto por aquellos elementos del sistema actual que será necesario modificar para crear el nuevo sistema. Además, será necesario indicar todos los nuevos elementos a introducir, así como la forma en que se integrarán en el sistema. Como el lector puede imaginarse, la realización del diseño de un nuevo sistema tiene muchos menos inconvenientes que el volver a "rediseñar" de un sistema ya existente.

Posiblemente, una de las herramientas más difundidas para la modelización sean los diagramas de flujos de datos

Una vez identificados los nuevos elementos del sistema, junto con las modificaciones que se realizarán, se procederá a realizar el diseño detallado. Como modelo se tendrán en cuenta los diferentes documentos generados en la fase de análisis, pero con las modificaciones que se han decidido. A partir de aquí, habrá que describir cada uno de los procesos con más detalle, de tal forma que se generen especificaciones para realizar los diferentes módulos de programas, se rediseñarán las bases de datos y los flujos de datos (se tendrán en cuenta el diccionario de datos y el modelo entidad-relación). Además, se habrán creado descripciones de los diferentes formularios y de las interfases de usuario. La normalización de todas estas tareas beneficiará el progreso de las fases de diseño y desarrollo, lo que redundará en una facilitación del trabajo de los grupos de programación.

DESARROLLO E IMPLEMENTACIÓN

Durante esta fase se procede a la construcción de las bases de datos, archivos y programas que reflejen las especificaciones producidas durante el proceso de diseño. También se procede a la prueba de los programas, así como al entrenamiento de los usuarios que lo utilizarán.

Por último, el nuevo sistema entra en funcionamiento. A partir de este momento, es posible que surjan algunos problemas, que deberán ser corregidos sobre la marcha.

SISTEMAS ABIERTOS

SERVIDORES CONCURRENTES

Fernando J. Echevarrieta

unix X

Numerosas han sido las peticiones de los lectores preguntando cómo realizar programas de comunicaciones con varios "canales" y cómo realizar comunicaciones entre más de dos procesos. Éstos serán los temas tratados en el presente artículo

Como habrán podido apreciar muchos lectores que, como es aconsejable, han pretendido ir más alla de los expuesto en los artículos anteriores, cuando se pretende realizar una aplicación distribuída, comienzan a surgir nuevas ideas y con ellas, nuevas necesidades. En ocasiones, no basta con un sólo cliente y un sólo servidor y, en la mayor parte de los casos, tampoco es suficiente con un único canal de comunicación. Con los dos primeros artículos correspondientes a la interfaz socket se sentaron las bases de esta técnica fundamental. En el presente artículo, con muy poco esfuerzo, se ampliarán horizontes hasta hacer realidad aquel slogan que decía "El límite es su imaginación".

SERVIDORES CONCURRENTES

El término concurrencia aplicado a procesos presenta una excesiva ambigüedad a pesar de tratarse de un término técnico. En el caso de las comunicaciones, se podría decir que un servidor es concurrente cuando atiende a dos o más clientes "a la vez".

Ya desde el punto de partida, cabe distinguir de forma drástica entre *concurrencia real* y *concurrencia aparente*. En el primero de los casos, es necesario un sistema multiprocesador debido a que un procesador no puede ejecutar más que un proceso a la vez. Sin embargo, se habla de concurrencia aparente cuando diversos procesos crean la ilusión de ejecutarse simultáneamente o, simplemente, los turnos de ejecución de los mismos no son apreciados por el proceso de información que se lleva a cabo. En el caso que nos ocupa, cada cliente tiene la "sensación" de ser el único atendido por el servidor.

En este segundo caso, también es posible realizar una división más sutil en dos niveles de concurrencia: un nivel será el que resulta del gestor de procesos del sistema operativo (*concurrencia multiproceso*) y otro el que los propios programas de aplicación logren simular (*concurrencia monoproceso*).

Son múltiples los motivos por los que resulta interesante el desarrollo de servidores concurrentes. El primero de ellos, el más obvio, es la programación sobre una plataforma multiprocesador. Si se divide un servidor en distintos procesos que atiendan, cada uno de ellos, a un cliente, es bastante probable que un sistema bien diseñado asigne cada uno a un procesador, con lo que se lograría maximizar el grado de concurrencia alcanzando una concurrencia real.

En el campo de la concurrencia aparente, otro de los motivos es que exista una gran diferencia entre los tiempos de respuesta necesarios para procesar las peticiones. Si se cuenta con un servidor iterativo, se atenderá a las distintas poblaciones según una disciplina *FCFS* (*First Come First Served*) mientras que si se desglosa el servidor en varios procesos, el gestor del S.O. los tratará según una disciplina *Round-Robin*, asignando "rodajas de tiempo" (*time slices*) por turnos a cada uno de ellos. En ambos casos, los resultados relativos a tiempos de espera en cola, estancia en el sistema, logitud de colas y otros estadísticos del sistema, variarán notablemente, aunque su cálculo y estimación es objeto de disciplinas como la *teoría de colas* y la *investigación de operaciones* y escapa al contenido de esta serie.

Por último, otro de los principales motivos es que el proceso de cada petición exija un amplio uso de E/S, por lo que el servidor podría atender otra petición mientras espera la terminación de una entrada o salida.

En definitiva, la intuición de "si se hace a la vez será más rápido" suele ser cierta en la mayoría de los casos aunque existe una excepción para confirmar la regla: el caso de servidores de datagramas.

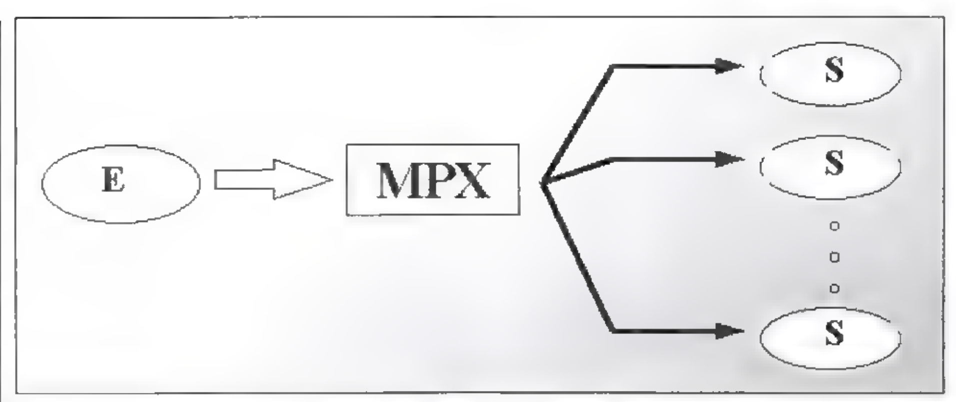
Al tratar con un servidor orientado a conexión, se recibe una petición de conexión, se acepta, se le asigna un socket esclavo y se genera un proceso que atienda a esa conexión. Así pues, se crea un proceso hijo por cada conexión y cada una de estas conexiones puede contar con un número indefinido de peticiones. Por tanto, se habla de *concurrencia entre conexiones*.

En el caso de un servidor de datagramas, es decir, no orientado a conexión, se generará un proceso por cada datagrama recibido. Por tanto, se trata de una *concurrencia entre peticiones*. Esto implica que existirá generación de procesos cada vez que se realice una petición en lugar de cada vez que se realice una conexión, como ocurría en el caso anterior. Por ello, el empleo de servidores de datagramas concurrentes y multiproceso suele ser bastante raro ya que sólo es aconsejable cuando el tiempo de creación de un proceso es notablemente inferior al de atención a una petición. Por otra parte en la mayoría de los casos en que se emplea un servicio de datagramas, los tiempos de atención suelen ser muy reducidos.

CONCURRENCIA MONOPROCESO VS MULTIPROCESO

En el caso de un servidor concurrente, debe ser posible aceptar varias peticiones "a la vez" y, por lo tanto, generar dos o más sockets esclavos para atenderlas.

Se habla de *servidores concurrentes multiproceso* cuando tras recibir una petición se genera un servidor esclavo para procesarla. De esta forma, el maestro sólo recibe peticiones y podrán existir varios esclavos atendiéndolas de forma concurrente, con una concurrencia gestionada por el S.O. Las ventajas de esta aproximación son múltiples: por un lado es más fácil ejecutar un programa compilado aparte para cada petición. Por otro lado, se trata de la solución más sencilla ya que el programador no tiene que gestionar el manejo de los distintos sockets porque de ello se encarga el S.O. De esta forma, cada servidor esclavo trata únicamente con un socket por el que realiza la comunicación.



Funcionamiento de un multirrepetidor.

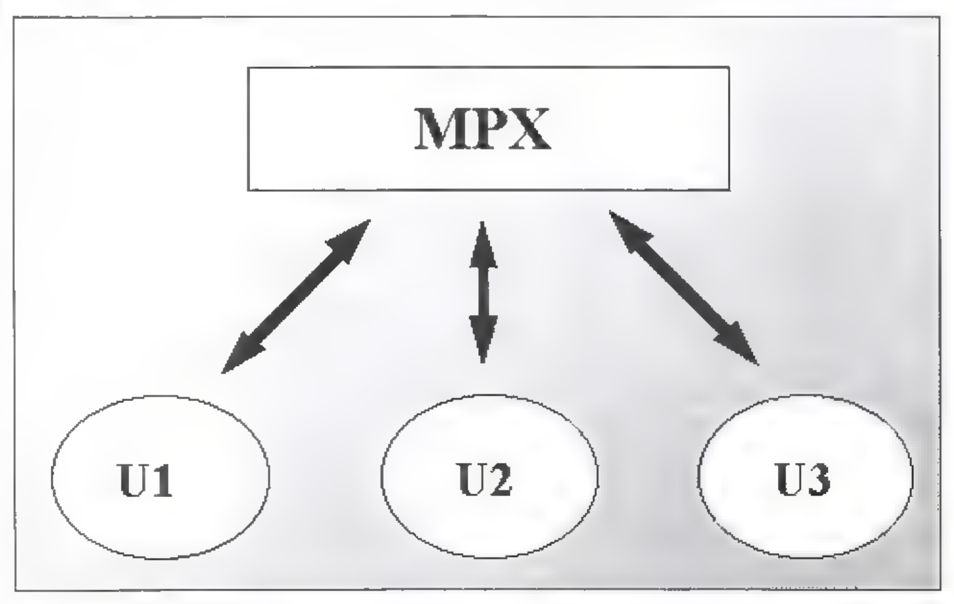
En el caso de un *servidor concurrente monoproceso*, la concurrencia aparente se logra realizando las operaciones E/S de manera asíncrona. Para ello, es necesario que el servidor maneje todas las conexiones a la vez. El procedimiento consiste en esperar a que se produzca una entrada o salida en cualquiera de las conexiones que tiene abiertas. En ese caso deberá atender a esa conexión y volver a esperar. El inconveniente es que el proceso es tan lento como un servidor iterativo. La ventaja, que las conexiones pueden compartir o intercambiar datos entre ellas.

Así pues, a la hora de optar por un servidor monoproceso o multiproceso, son relevantes tres factores de decisión:

1. La necesidad de compartir o intercambiar datos entre las conexiones atendidas, lo que impone un servidor monoproceso. Este es el ejemplo de multirrepetidores, multiplexores o sistemas de comunicación multiusuario.
2. El tiempo de respuesta a una petición. Cuanto más alto sea, mayor será la necesidad de alejarse del modelo iterativo, lo que conduce a un sistema multiproceso.
3. La existencia de un hardware multiprocesador, que sugiere el uso de servidores multiproceso.

SERVIDORES CONCURRENTES MULTIPROCESO

Por las razones expuestas en el primer punto, se supondrá que un servidor concurrente monoproceso será orientado a conexión, aunque la forma de pro-



Multirrepetidor simétrico.

ceder sería análoga en un servidor de datagramas. Para realizarlo se deben seguir los siguientes pasos:

Servidor maestro:
1. Creación de un socket (*socket*)
2. "Enganche" al puerto especificado para el servicio (*bind*)
3. Puesta a la escucha (*listen*)
4. Admisión de conexiones (*accept*)
5. Generación de un servidor esclavo (*fork*)

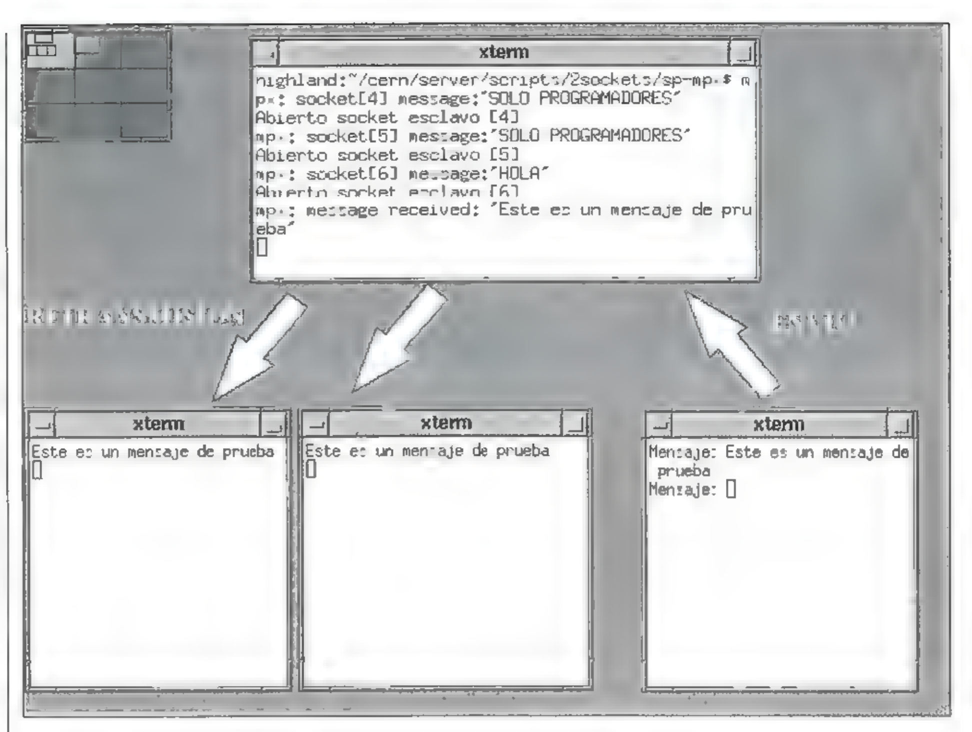
Al llegar a este punto, maestro y esclavo, compartirán una copia de los sockets por lo que lo primero que debe hacer el servidor maestro es cerrar su copia del socket esclavo y el servidor esclavo realizar lo propio con su copia del socket maestro. Tras ello, el servidor maestro volverá al punto 4 y el esclavo continuará con los pasos:

Servidor esclavo:
0. Como se mencionaba en el punto anterior, es bastante simple ejecutar un programa compilado aparte para atender la petición. Si este paso opcional se realiza, el servidor esclavo realizará una llamada *execve* para ser sustituído por el otro programa.
1. Lectura de la petición (*read*)
2. Atención a la petición y posiblemente, respuesta (*write*)
3. Cierre de la conexión (*close*)
4. Salida (*exit*)

Obsérvese que todo servidor esclavo debe morir tras realizar su función, lo que no afectará al servicio ya que que el servidor maestro seguirá vivo para recibir nuevas peticiones y generar nuevos esclavos para procesarlas.

LA LLAMADA AL SISTEMA SELECT

La llamada al sistema *select* proporciona E/S asíncrona permitiendo a un proceso que espere al primer descriptor activo de entre un conjunto de descriptores. Este es el fundamento de los servidores concurrentes monoproceso. Su versatilidad se encuentra algo reñida con su facilidad de uso ya que no es precisamente de las llamadas al sistema más sencillas de manejar de la API de UNIX. La forma que adopta es:
select(num_desc,r_desc,w_desc,x_desc,time_out)



Multirrepetidor del artículo funcionando sobre una plataforma Linux.

La idea de esta llamada consiste en realizar un conjunto con los descriptores para los que se espera una entrada, otro con los que se espera una salida y un tercero con aquellos en los que se espera una excepción. Al ejecutar *select*, el sistema, en principio, queda a la espera de uno de estos eventos y, cuando se produce uno, modifica los valores de los conjuntos indicando los descriptores de los ficheros que han sido activados.

Así pues, la declaración y significado de los parámetros es el siguiente:
num_desc: entero que indica el número máximo de descriptores a ser considerados.
r_desc, *w_desc* y *x_desc*: punteros a un tipo *fd_set*, conjunto de descriptores para los que se espera entrada, salida o excepción respectivamente.
time_out: estructura *timeval* de la forma:

```
struct timeval {
  long tv_sec;
  long tv_usec;
};
```

Donde *tv_sec* es un tiempo en segundos y *tv_usec* en microsegundos. Sin este parámetro la llamada sería bloqueante, con él, se puede definir un tiempo de espera máximo a que retorne la llamada.

Todo esto sería maravilloso si C definiera un tipo de datos conjunto y operaciones sobre él. Sin embargo, no es así. El tipo *fd_set* es, en realidad un tipo entero en el que el valor del bit i-ésimo indica el estado del descriptor número i. Por fortuna, se han definido una serie de macros para operar sobre este tipo como si de un tipo conjunto se tratara:
FD_SET: Añade un descriptor a un conjunto
FD_CLR: Elimina un descriptor de un conjunto
FD_ZERO: Pone a cero todo el conjunto
FD_ISSET: Devuelve 0 si el descriptor no pertenece al conjunto u otro valor si pertenece.

Para utilizar esta llamada es necesario incluir
#include <sys/time.h>
donde se definen las macros y la estructura *timeval*.
#include <sys/types.h>
donde se define el tipo *fd_set*
así como
#include <unistd.h>

UNA NOTA SOBRE PORTABILIDAD

En linux, al retornar, *select* modifica el valor de la estructura para indicar el tiempo que resta para agotar el *time*



LISTADO 1

```
main()
{
        int i;

        char message[MAXLENMSG];

        /* Paso a background */
        if ( fork() != 0 ) {
          printf("mpx: OK\n");
          exit(0);
        }

        /* Comienza proceso de comunicacion
*/
        client   = launch();

        num_sock_set = getdtablesize();
        FD_ZERO(&active_sock_set);
        FD_SET(client,   &active_sock_set);

        while (1) {
            bcopy((char *)&active_sock_set,(char
*)&sock_set,sizeof(sock_set));


select(num_sock_set,&sock_set,(fd_set
*)0,(fd_set *)0,
             (struct timeval *)0);

       if (FD_ISSET(client, &sock_set)) {
          ns=create_slave_sock(client);
                   printf("Abierto socket esclavo
[%d]\n",ns);
          FD_SET(ns, &active_sock_set);
       }

             /* Tres primersos desc son STD
IN/OUT/ERR */
      for (file_desc=3; file_desc < num_sock_set;
++file_desc)

       if ((file_desc != client) &&
             FD_ISSET(file_desc,&sock_set))

             if (client_process(file_desc) == 0) {
                close(file_desc);
                   printf("Cerrado socket esclavo
[%d]\n",file_desc);
                                  FD_CLR(file_desc,
&active_sock_set);
             }
        }
}
```

Función main del multirrepetidor simétrico.

out. Por ello, si no se restaura el valor original en cada llamada a *select*, el valor irá disminuyendo, por lo que puede llegar a tener una llamada no bloqueante. Esta característica es dependiente de la versión del sistema operativo. En otros sistemas, esto no ocurre. Éste es uno de los indeseables casos en que el programa compila perfectamente pero, en unos casos sí y en otros no, funciona de forma extraña. Por ello, y con el fin de asegurar la portabilidad, conviene controlar el valor de *time_out* siempre inmediatamente antes de realizar la llamada.

SERVIDORES CONCURRENTES MONOPROCESO

Una vez se dipone de la llamada *select*, este tipo de servidores se realiza mediante los siguientes pasos:

1. Creación de un socket (*socket*)

2. "Enganche" al puerto especificado para el servicio (*bind*).

3. Espera de E/S en los descriptores existentes (*select*)

4. Si el socket maestro está listo:
a) Admisión de la conexión y generación de un nuevo socket esclavo (*accept*)
b) Inclusión del nuevo socket en el conjunto de descriptores a la espera

5. Si un socket distinto del maestro está listo:
a) Lectura de la petición (*read*)
b) Atención a la petición y posiblemente, respuesta (*write*)

6. Vuelta al punto 2.

MULTIRREPETIDORES

Como ejemplo sencillo de servidor concurrente monoproceso basado en *select* se pondrá un ejemplo de un simple multirrepetidor orientado a conexión. El funcionamiento de un multirrepetidor es muy simple: se trata de un sistema con una entrada y múltiples salidas en las que reproduce todo que se le presenta en la entrada.

Como ejemplo, se va a desarrollar un multirrepetidor simétrico en el que no existe diferencia entre entradas y salidas. Así, cualquier cosa que se reciba por cualquiera de las conexiónes será retransmitida a todas las demás. El servidor escuchará en un puerto conocido: el 2001, aceptará un número indefinido de conexiones simultáneas y esperará en todas a la vez la recepción de algun mensaje. Cuando lo reciba procederá a retransmitirlo por todas las conexiones abiertas. Para el ejemplo se han desarrollado dos clientes, uno que envia datos denominado *e-send* y otro que los recibe denominado *client*.

Este ejercicio tiene múltiples aplicaciones prácticas además de la puramente didáctica. En este sistema están

LISTADO 2

```
client_process(int socket)
/* Processes a client request */
{
      char buf[256];
      int c;

      if ((c=read(socket,buf,MAXLENMSG)) > 0) {
              buf[c]='\0';
                     printf("mpx:  message  received:
'%s'\n",buf);


        for (file_desc=0; file_desc < num_sock_set;
++file_desc)

             /* Retransmision to all slave sockets */
             if ((file_desc != client) &&
              FD_ISSET(file_desc,&active_sock_set))
                 write(file_desc,buf,c);
      }


      return c;

}
```

Función de atención del multirrepetidor simétrico.

basados los protocolos de difusión (aunque se realizan en niveles inferiores): el servidor podría ser una máquina que conectara varias subredes.

Si existe un cliente que emite y muchos que reciben, se consigue un efecto de difusión similar al de la radio o la televisión. Si, en cambio, los clientes son capaces de emitir y recibir simultáneamente, se conseguirá un mecanismo de comunicación bidireccional y multipunto, es decir, un foro virtual en el que todos los usuarios son capaces de leer lo que todos los demás escriben.

CÓDIGO DEL MULTIRREPETIDOR SIMÉTRICO

El listado 1 muestra el código de la función *main()* del programa, donde reside el núcleo del servidor. Como suele ser habitual comienza pasándose automáticamente a background, tras lo cual se convierte en servidor mediante el procedimiento *launch()* que genera un socket maestro por el que recibir las peticiones de los clientes y al que se ha llamado *client*. El procedimiento *launch()* no debería

plantear ya ninguna duda al lector, que puede encontrar toda la información al respecto en números anteriores.

Tras ello, se emplea la llamada al sistema *getdtablesize()* para preguntar al S.O. el número máximo de descriptores de fichero que se pueden abrir simultáneamente. Este valor se almacena en *num_sock_set* usada con el *select*.

El servidor espera a recibir conexiones, por lo que *select* únicamente deberá esperar por entradas en un conjunto de descriptores. Así pues, no es necesario definir conjuntos para salidas y excepciones. Sin embargo, *select*, por cada tipo de evento a esperar, recibe un conjunto de descriptores y cuando el evento se produce, modifica el conjunto, que pasa a mostrar aquellos descriptores que se han activado. En este paso, se pierde la información que contenía el conjunto original. Por ello, será necesario definir dos conjuntos de descriptores por cada tipo de evento: uno para almacenar la información de los descriptores que se desea estudiar, al que se denominará *descriptores activos*, y otro para utilizar como intercambio con el *select*. En el programa se les ha denominado *active_sock_set* y *sock_set* respectivamente. En caso de que no sólo se atendiera a entradas sino también a salidas y excepciones, se procedería de forma análoga definiendo un par de conjuntos para cada uno de estos tipos de eventos.

El conjunto de descriptores activos debe contener al inicio únicamente el descriptor de socket maestro *client*. Por ello, primeramente se pone a cero:

FD_ZERO(&active_sock_set)

y después se indica el socket como activo:

FD_SET(client,&active_sock_set)

Con ello, finaliza la parte de preparación y comienza el bucle sin fin que supone el servidor. Para ello, se realiza una copia del conjunto *active_sock_set* en *sock_set* con el fin de facilitársela al *select*. Con este fin se ha usado en el ejemplo la llamada *bcopy*.

El siguiente paso es, por fin, la llamada *select* a la que se ha llamado sin *time out* (bloqueante) y pasándole conjuntos vacíos de descriptores para salida y excepciones.

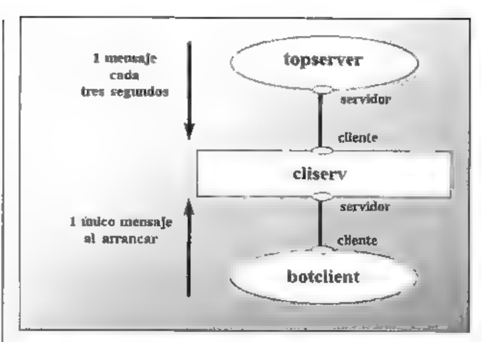
Cuando la llamada retorne, habrá modificado el valor de *sock_set*, lo que no debe preocupar ya que, como se justificaba anteriormente, se conserva el conjunto de conexiones abiertas en la otra variable: *active_sock_set*.

Primeramente se estudia si se encuentra activo el socket maestro mediante la macro *FD_ISSET*. Si así ocurre, es que se ha solicitado una nueva conexión, en cuyo caso se genera un socket esclavo *ns* para atenderla mediante el procedimiento *create_slave_sock (listado 2)*. Este socket debe ser incluído en el conjunto de descriptores activos para que la próxima llamada a *select* lo "vigile" también a él. Para ello, se emplea la macro *FD_SET*. El procedimiento *create_slave_sock* se ha realizado conforme a lo explicado en artículos anteriores.

Tras ello, se pasa a revisar si alguno de los sockets esclavos se encuentra también activo. Esta información se encuentra en *sock_set*. No será necesario que el programa genere una lista con los sockets que tiene abiertos debido a que si un socket no estaba incluído en *active_sock_set* no aparecerá en *sock_set*. Así pues, para realizar esta verificación se puede probar con todos los números enteros. En realidad, se sabe a priori que los tres primeros (partiendo del cero) corresponden a entrada y salida estandar y salida de error respectivamente. Así pues, se puede acotar la revisión a un rango entre el cuarto descriptor y el número máximo *num_sock_set* calculado anteriormente. Para ello, se emplea de nuevo la macro *FD_ISSET* excluyendo de la comparación el socket maestro que ya se ha gestionado anteriormente. Cuando uno de los sockets esclavos se encuentra activo, el procedimiento *client_process* se encarga de atenderlo. Cuando se ha terminado por completo de atender una conexión (el procedimiento devuelve 0), el socket se cierra y se procede a eliminarlo del conjunto de descriptores activos mediante *FD_ZERO*.

Una vez se ha revisado el estado de los sockets maestro y esclavos se vuelve a realizar una copia de *active_sock_set* en *sock_set* y el bucle comienza de nuevo.

Como se puede apreciar, el funcionamiento del servidor equivale al de uno iterativo: se atiende secuencialmente las peticiones de las conexiones. Sin embargo se produce una concurrencia de conexiones al permitir tener abiertas un número indefinido de las mismas y atenderlas todas "a la vez"



Esquema de la aplicación cliente y servidora del ejemplo.

COMUNICACIONES "MULTICANAL"

Un bonito y didáctico ejercicio que se propone al lector es fundir los dos tipos de cliente que se han realizado en la aplicación ejemplo en uno sólo que sea capaz de enviar y recibir simultáneamente. Para ello, el cliente deberá atender concurrentemente las comunicaciones con la red y con el teclado. Sin embargo, esto no introduce mayor complejidad. A lo largo del texto se ha cuidado de emplear el término *descriptor de fichero* y no *descriptor de socket*. Recuerde el lector que para UNIX el universo está constituido por ficheros y que los descriptores de socket se toman del mismo conjunto que los del resto de ficheros. Así pues, basta introducir el descriptor de teclado, o el de la entrada estándar, en el conjunto de descriptores activos que vigila el *select* y darle el mismo tratamiento que a un socket. De esta forma generará un programa de comunicaciones como el *talk* pero multiusuario, un foro virtual como el que se ha sugerido con anterioridad.

Otra de las posibles mejoras consite en dotar de cierta "inteligencia" al multirrepetidor. Piense en la posibilidad de asociar conexiones entre sí, lo que equi-

En ocasiones no es suficiente con una aplicación con un único cliente y un único servidor

CONCURSO DE PROGRAMACION DE COMUNICACIONES SOBRE UNIX

SÓLO PROGRAMADORES convoca el Primer Concurso de Programación de Comunicaciones sobre UNIX. Para concursar, envie el código fuente de sus programas en un disquete a la redacción de la revista acompañado de todos sus datos personales y, si dispone, de su dirección de correo electrónico.

Las aplicaciones deberán compilar sobre Linux o SunOS y hacer uso de la interfaz de sockets (tratada en los números 17, 18 y 21) o de RPCs de SUN (números 19 y 20). Los programas deberán ser totalmente originales o estar basados en el código de los ejemplos presentados en el curso. Por ejemplo, se sugiere realizar alguna de las mejoras a las que se hace referencia en el apartado "COMUNICACIONES MULTICANAL" del artículo del número 21. TODOS los programas deben ir acompañados de un fichero Makefile de tal modo que para su compilación baste teclear *make*.

Se valorará por igual:

1. La originalidad: ideas nuevas, curiosas o sorprendentes.
2. La posible conexión a servicios existentes: clientes o servidores de protocolos habituales con o sin mejoras, pasarelas entre servicios, etc.
3. El diseño del sistema de comunicaciones: diseño orientado a la comunicación (protocolos) o a la aplicación.
4. La claridad y el estilo.
5. La documentación aportada.
6. La sencillez.

Las mejores aplicaciones serán publicadas en uno de los próximos CD-ROMs que habitualmente acompañan a la revista. Además, el ganador recibirá una suscripción gratuita por un año a la revista, además de un pack de libros de la colección Biblioteca Profesional de la editorial Tower Communications.

valdría a generar diferentes "canales" de comunicación. Podría permitirse el salto de un canal a otro desde las propias conexiones si el servidor analizara su contenido en busca de comandos, con lo que se habría construido un sistema similar al *IRC* (*International Relay Chat*). Otra posibilidad sería crear un control para el multirrepetidor, con lo que actuaría como una central de conmutación. En definitiva, este es un punto del recorrido en el que se abre multitud de nuevos caminos, disfrute recorriéndolos y aproveche para concursar con SÓLO PROGRAMADORES (ver cuadro).

APLICACIONES CLIENTE Y SERVIDOR

En la mayor parte de las ocasiones, una aplicación que consta de un único cliente y un único servidor no es suficiente. En algunos casos, una aplicación está relacionada con varios servicios, siendo cliente de un servicio y servidora de otro. En otros, en los que el grado de distribución es mayor, puede incluso ser cliente y servidora del mismo servicio, como ocurre con los servidores proxy. En ambos casos, el diseño de la aplicación no sufrirá grandes cambios. Como es de suponer, constará de código cliente y código servidor y prestará atención en un puerto mediante un socket a la vez que emplea otro para conectarse a un servidor. En estos casos, se trata, una vez más, con un tipo de concurrencia monoproceso, por lo que será necesario emplear de nuevo la llamada *select*.

Como ejemplo se ha desarrollado una aplicación que hace las funciones de cliente y servidor simltáneamente a la que se ha llamado *cliserv*. Actúa como cliente de otra aplicación denominada *topserv* que genera mensajes periódicamente empleando señales (que se tratarán en un próximo artículo) y como servidora de una serie de clientes, *botclient*. Cuando se ejecuta un cliente *botclient*, se conecta a *cliserv* y envía un mensaje, *cliserv* contesta y después se cierra la conexión. La aplicación *cliserv* muestra en pantalla trazas de los mensajes recibidos.

El funcionamiento de la aplicación es completamente análogo al del multirrepetidor simétrico descrito anteriormente, por lo que huelgan más explicaciones.

SOFTWARE FACILITADO

Las fuentes del servidor multirrepetidor simétrico han sido incluídas en el disco de la revista comprimido en el fichero *sp-mpx.tgz*.

Las fuentes de la aplicación cliente y servidora simultáneamente han sido incluídas comprimidas en *sp-2sock.tgz*.

Para la extracción de los ficheros basta teclear:

tar xvfz <fichero>.tgz

Si en su sistema operativo no cuenta con el tar de GNU, la extracción deberá realizarse en dos pasos:

gunzip <fichero>.tgz

tar xvf <fichero>.tar

Con las aplicaciones se ha incluído un Makefile por lo que para la generación de los ejecutables bastará teclear *make*.

En linux para eliminar los demonios que actúan en background bastará teclear *make mata*. En otros sistemas habrá que eliminarlos manualmente.

CONCLUSIÓN

Con el presente artículo se ha completado la visión de las arquitecturas cliente-servidor orientadas a conexión y basadas en la interfaz socket. Aún quedan algunos aspectos técnicos importantes por considerar que, si bien no afectan a la arquitectura, sí son importantes desde el punto de vista de la implementación, como son el acceso a bases de datos de nombres, direcciones y servicios. Con estás técnicas que se introducirán el mes próximo, dejará de ser necesario "cablear" direcciones y puertos en el código de los programas y se podrá acceder a la información de sistemas como el DNS, NIS, etc. Al tiempo que se estudian desde el punto de vista de la programación, se comenzará a tocar el tema de la administración de sistemas UNIX comenzando por estas bases de datos de red.

CONTACTAR CON EL AUTOR

Para cualquier clase de duda, aclaración, comentario, sugerencia o crítica, se anima al lector a que se ponga en contacto con el autor a través de carta a la redacción de la revista o, preferiblemente, mediante:

E-Mail Internet: echeva@dit.upm.es
E-Mail Compuserve: 100646,2456
WWW: http://highland.dit.upm.es:8000
ftp: highland.dit.upm.es



FICHEROS EN MVS (III)

José María Peco

Para terminar el tema ficheros en MVS, este mes se dedica esta sección a estudiar la organizacion VSAM, la cual da servicio a un tipo de ficheros muy importante, nacido en los años 60, antes de la aparición de los sistemas gestores de Bases de datos, pero que a pesar de su edad, no sólo mantiene totalmente su vigencia, sino que sigue jugando un papel muy importante en cualquier GRAN SISTEMA

En las entregas anteriores se han enumerado los distintos métodos de almacenamiento y acceso a la información bajo el S.O. MVS de IBM. También se hizo una introducción a la memoria virtual, por lo que el presente articulo se limita, simplemente, a la organizacion *VSAM* (*Virtual Storage Access Method*).

En esta organización, dejan de tener importancia los términos de *cilindro*, *cabeza* o *pista*, vistos en los artículos anteriores. En cambio, aparecen otros conceptos que sirven para referenciar un registro. Éstos son los de *RBA* (*Relative Byte Address*) e *IC* (*Intervalo de control*).

VSAM es una organizacion de datos que admite registros de longitud fija y variable. Realmente, este método está pensado para optimizar el almacenamiento liberando al programador de la gestión que requiere el uso de registros de longitud variable, al tiempo que gestiona el bloqueo y desbloqueo de registros, calculando el factor de bloqueo, a fin de aprovechar al máximo el espacio del dispositivo.

OORGANIZACIÓN DE LOS FICHEROS VSAM:

Existen tres tipos de ficheros:

- **ESDS** (*Entry Sequence Data Set*): Equivalentes a ficheros secuenciales.
- **KSDS** (*Key Sequence Data Set*): en secuencia de clave. Este tipo de fichero es muy parecido al de organización secuencial indexada, teniendo la información dividida en dos partes: Área de índices y Área de datos.
- **RRDS** (*Relative Records Data Set*): equivalentes a ficheros con organizacion directa.

ESTRUCTURA FÍSICA DE FICHEROS VSAM:

Como se acaba de decir, VSAM libera al informático de hacer consideraciones sobre cilindros y cabezas (direcciones normales en disco) para dejar que el programador piense que el espacio de disco donde se graban los registros es continuo, comenzando en la dirección cero. Este espacio en disco, que puede abarcar uno o varios volúmenes, se denomina *DATA SPACE*, y se define mediante la etiqueta correspondiente en la *VTOC*, ya que, al fin y al cabo, es un fichero más del sistema.

En caso de que, por su tamaño, abarcara varios volúmenes, éstos deberán ser del mismo tipo de dispositivo, ya que considera todo su espacio como homogéneo en cuanto a características. Y, como ya se dijo en el primer artículo, cada tipo de disco tiene sus propias características, tales como longitud de pista, numero de pistas por cilindros, etc. Por otra parte, y como es lógico, en un mismo dispositivo pueden existir uno o varios *DATA SPACEs* junto con cualquier otro tipo de organizacion de datos.

CONCEPTOS ESPECÍFICOS PARA VSAM

- *Registro Lógico*: Es la unidad básica de datos, es decir, es la información que será tratada por el informático dentro del programa. Admite registros de longitud fija y variable.
- *Registro Almacenado*: Es un registro lógico mas un campo de control que se denomina *RDF* (*Record Definition Field* o campo de definición de registro). Este campo de control, *RDF*, consta de tres octetos (Figura 1):
- byte 1: contiene indicadores de control, teniendo significado los bits 1 y 4.

Valores y significado de:

- bit_1 = 0 indica que no sigue más información de control relacionada con el *RDF*
- bit_1 = 1 indica que sigue más información de control relacionada con el *RDF*.
- bit_4 = 0 indica que el descriptor del registro es único (sólo hay un registro con esa longitud)
- bit_4 = 1 indica que es un descriptor de cuenta, es decir, indica el numero de registros que tienen la misma longitud.

La figura 3, que se comenta posteriormente, muestra el uso de este campo de control.

Intervalo de control: El espacio total ocupado por un fichero VSAM está dividido en áreas que se denominan intervalos de control. VSAM almacena los registros de datos y la información de control que los describe en estas áreas (figura 2).

Un registro de datos, junto con la información de control, es lo que configura el *Registro Almacenado*. Todo intervalo de control lleva adicionado un campo de control que se denomina *CIDF* (*Control Interval Description Field*).

El intervalo de control puede variar de un fichero a otro, pero dentro de un fichero, la longitud del intervalo de control permanece constante, es decir, a lo largo de un fichero, la longitud de los intervalos de control no varia. VSAM elige ésta longitud teniendo en cuenta lo siguiente:

- debe ser múltiplo de 512
- longitud de los registros de datos
- tipo de *DAAD* (*Dispositivo de Almacenamiento de Acceso Directo*)
- Memoria disponible
- Tamaño de las *IOAREAS* facilitado en la creación del fichero.

CIDF: es el campo de definición del intervalo de control. Tiene 4 octetos de longitud, de los cuales, los 2 primeros indican el desplazamiento en octetos desde el principio del intervalo de control a las áreas de espacio libre; los dos siguientes indican el tamaño del área de espacio libre contenida en el intervalo de control, pues VSAM intenta dejar espacio libre en los intervalos de control para las adiciones.

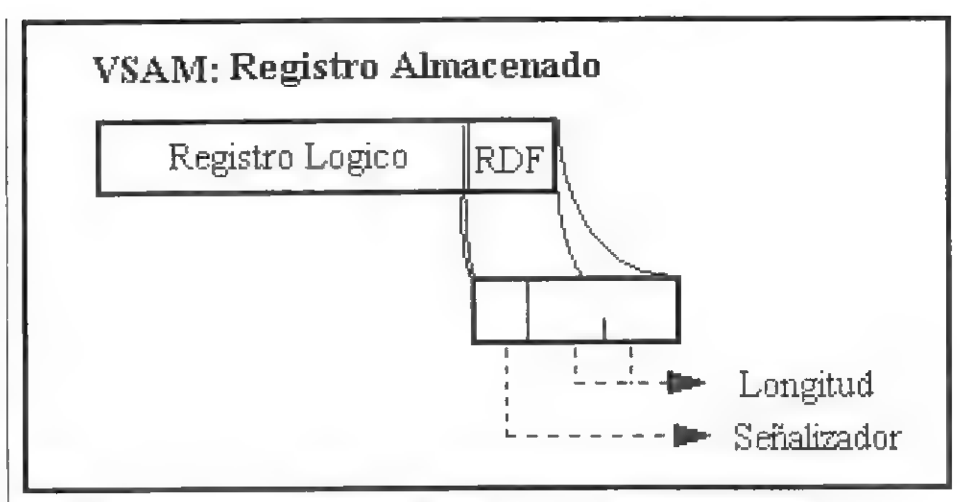


Fig1.- Contenido del campo de control RDF.

EJEMPLO

La figura 3 muestra el contenido de los *RDF* que acompañan a los registros lógicos grabados en el intervalo de control de la figura 2.

Como el registro lógico 1 va seguido por un registro de distinta longitud, utiliza sólo un *RDF*.

Como los registros lógicos 2, 3 y 4 tienen la misma longitud, se utilizan 2 *RDF* para describir a los 3 registros lógicos, con lo que se ahorra espacio. El primer *RDF* tiene un 1 en el bit 4 para indicar que la información contenida en los dos bytes siguientes especifican, en binario, el numero de registros que tienen esa longitud (11 (binario)=3(decimal)). El segundo *RDF* contiene 0 en los bits 1 y 4 pues no sigue más información de control; en los dos bytes finales se encuentra la longitud del registro (1100100=100).

Área de control: Recibe esta denominación cada uno de los conjuntos formados por un numero entero de intervalos de control. Este numero lo fija el propio VSAM, siendo como mínimo de tres. Un área de control ocupa un número entero de pistas, generalmente llegando a un cilindro, pero nunca mayor de un cilindro. Así, en el caso de elegir 100 intervalos de control por área, los primeros 100 intervalos constituyen la primera área de control, los 100 siguientes la segunda, etc.

Por áreas de control, VSAM reserva intervalos de control libres para adiciones y reorganizaciones (figura 4).

Cluster: En VSAM, recibe el nombre de *cluster* el *data space*, el cual contiene:

en *KSDS*: las áreas de datos y las áreas de índices

en *ESDS*: el área de datos

en *RRDS*: el área de datos.

RBA: dirección de byte relativa a origen de fichero

Para direccionar registros, VSAM no tiene en cuenta la ubicación física del registro en el dispositivo (concepto cabeza cilindro), sino que VSAM direcciona mediante el desplazamiento en octetos del registro respecto al comienzo del espacio reservado al fichero. Este desplazamiento se denomina *RBA* (*Relative Byte Address*).

Para el cálculo de los *RBA*, VSAM considera tanto registros de datos, como espacios libres y campos de control (figura 4).

VSAM es una organizacion de datos que admite registros de longitud fija y variable

TIPOS DE FICHEROS VSAM

- *ESDS - Fichero en secuencia de entrada.*

En este tipo de ficheros, los registros se graban en el mismo orden en el que van llegando, es decir, uno detrás de otro, sin dejar huecos libres. Los regis-

tros nunca se intercalan, y tampoco se podrán ni acortar ni alargar. Las adiciones sólo podrán hacerse al final del fichero por ser una organizacion compacta.

Cuando se quiere borrar un registro, se marcará físicamente, pero se eliminara físicamente cuando se realice la posterior reorganización.

Los *RBA* no varían. Los registros siempre tienen el mismo *RBA*. El acceso a lo registros será secuencial, aunque también podrá hacerse al azar, conociendo su *RBA*, para lo cual basta con leer previamente el registro y recoger así el *RBA*, pues VSAM no mantiene ningún tipo de índice con este tipo de fichero.

- *KSDS : Fichero en secuencia de clave.*

Se corresponde con la organizacion secuencial indexada. Este tipo de fichero tiene almacenado sus registros en secuencia de clave llevando un orden lógico. Para poder realizar su cometido, un fichero de este tipo va siempre acompañado de un índice que se usa para localizar los registros de datos. Los ficheros *KSDS* están formado por dos partes: Índice y Espacio de datos, y no tendrá área de *overflow* porque VSAM reserva áreas libres en los intervalos de control.

ÍNDICES

Los índices tienen por finalidad el relacionar las claves con la posición de los registros de datos en el fichero. Será VSAM el que genere y mantenga este conjunto de índices.

Los índices se agrupan a su vez, manteniendo la estructura de un fichero, con intervalos y áreas de control. Cada intervalo de control recibe el nombre de *registro*, y contiene varias *entradas* (figura 5).

El índice puede tener uno o más niveles. Cada uno de los niveles está compuesto por un conjunto de registros que contendrán una serie de entradas con la dirección de los registros de índices de jerarquía inmediatamente inferior.

El índice de mas bajo nivel recibe el nombre de *Conjunto de secuencia* (*sequenced-set*). Cada entrada de este conjunto de secuencia apunta a los intervalos de control que contienen los registros de datos en el fichero.

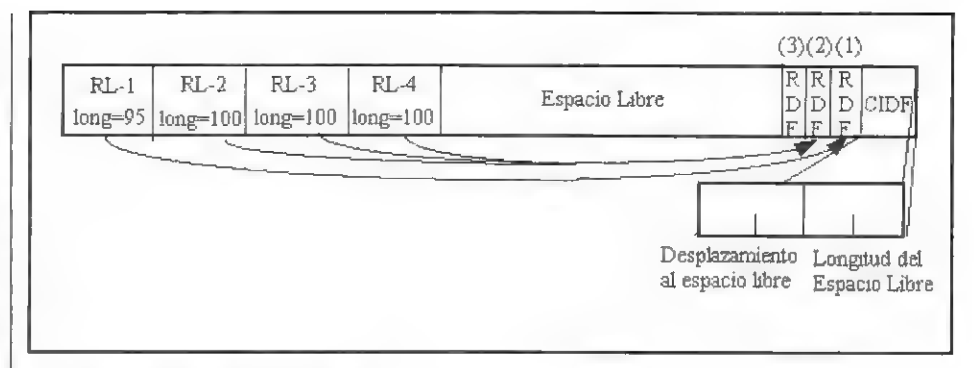


Fig 2.- VSAM: Intervalo de control.

Cada *entrada* de un registro índice contiene:

- La *clave* más alta de cada uno de los registros del nivel inferior de índices; y
- Un *puntero* que se podría llamar vertical, para indicar la dirección de ese registro, es decir, es parecido al campo de enlace de la organización secuencial indexada.

Esta organizacion se aprecia en todos los niveles, hasta llegar al *conjunto de secuencia* que es el nivel más bajo de la jerarquía de índices.

Cada uno de los *registros* del *conjunto de secuencia* remite a un *área de control*; y

Cada una de las *entradas* de un registro del *conjunto de secuencia*, remite a un *intervalo de control*.

Es decir, si en un fichero, un *área de control* tiene 8 intervalos de control, un registro *del conjunto de secuencia* tendrá 8 entradas.

- Los *intervalos de control* libres también están direccionados en el *conjunto de secuencia*.

El *conjunto de secuencia* relaciona a todos los registros por punteros horizontales para poder realizar un proceso secuencial, tal y como indica la figura 5.

- VSAM actualiza y reorganiza los índices cuando lo cree necesario.

ÁREA DE DATOS

Los registros con datos de usuario se graban en intervalos de control, siguiendo una secuencia lógica. Los intervalos de control se agrupan en áreas de control, y éstas estarán referenciadas de alguna manera en cada uno de los intervalos de control del conjunto de secuencia, como ya se ha visto.

ÁREA DE ESPACIO LIBRE

En los ficheros *VSAM-KSDS* se puede dejar espacio libre para añadir, alargar o acortar registros de datos. Estas áreas de espacio libre pueden ser de dos tipos:

- Espacio libre al final de los intervalos de control utilizados para datos.
- Intervalos de control libres. Varios en cada área de control.

El tamaño y el modo de dejar el espacio libre es facilitado por el propio programador en el momento de la creación, siendo VSAM el que se encargue de utilizarlo adecuadamente.

En un mismo dispositivo pueden existir uno o varios DATA SPACEs junto con cualquier otro tipo de organizacion de datos

DIVISIÓN DE LOS INTERVALOS DE CONTROL

Cuando al añadir o alargar un registro se exceda de la capacidad de un intervalo de control, VSAM utiliza un intervalo de control libre que se encuentre dentro de la correspondiente área de control y reparte con él el contenido del intervalo de control excedido, quedando ambos con cierta cantidad de espacio libre.

En el caso de que el área de control se quede sin intervalos de control libres, VSAM generara una nueva área de control al final del fichero y repartirá con ella el contenido de esta.

Cuando se reparten áreas de control, el área de datos no lleva ni secuencia lógica ni física, la secuencia lógica será llevada por el conjunto de secuencia

PROCESOS DE UN FICHEROS VSAM

- *Carga*: La carga se suele realizar de forma secuencial. Se dejará espacio libre dentro de los intervalos de control y también intervalos de control libres dentro de las áreas de control.
- *Adiciones*: Con las adiciones se pueden dar los tres casos siguientes.
- Se podrán hacer adiciones de registro sin que se pierda la secuencia lógica y física, gracias a los espacios libres dejados en los intervalos de control en la creación.
- Cuando se produce la división de los intervalos de control dentro de la misma área de control, puede perderse la secuencia física, pero nunca se pedrera la lógica gracias a los punteros del conjunto de secuencia.
- Adiciones repartiendo áreas de control: entonces se dividen las áreas de control y se pierde la secuencia física pero nunca la lógica.
- *Supresiones*: Los registros se borran físicamente y se condensarán todas las informaciones que se encuentran en el intervalo de control, de forma que el espacio ocupado por ese registro, que es baja, pasa a ser adicionado al espacio libre actualizándose los índices correspondientes.

Como consecuencia de estas adiciones, divisiones y supresiones, los *RBA* de los registros se alteraran.

- *Recuperación*: Puede ser secuencial o al azar.
- *Secuencial*: VSAM sólo utiliza el conjunto de secuencia, utilizando los punteros horizontales para poder mantener la secuencia.
- *Azar*: VSAM va siguiendo todos los niveles de índices mediante los punteros verticales para localizar el intervalo de control del registro pedido.

Byte		------1-----	------2------	-----3------
bit		0123-4567	0123-4567	0123-4567
RDF	(1)	0000-0000	0000-0000	0101-1111
	(2)	0100-1000	0000-0000	0000-0011
	(3)	0000-0000	0000-0000	0110-0100

Fig 3.- RDFs de la figura 2.

RRDS : Fichero de registro relativo

Esta organización podría asimilarse con la Organización Directa Relativa vista anteriormente.

Los ficheros RRDS podrá actualizar, borrar y añadir registros en los huecos libres

Este fichero consta de una serie de huecos o registros, todos de igual longitud, y que, de alguna forma, se encuentran numerados correlativamente desde el primero de ellos. Se podría comparar con una tabla o array, donde cada registro de datos ocupa un único hueco y se carga o recupera con respecto a su posición relativa a origen de fichero.

Un fichero *RRDS* podrá actualizar registros, borrarlos o adicionarlos en los huecos libres.

Hay veces que se puede obtener a partir de una clave la dirección relativa a origen, aplicando una función de direccionamiento.

CATÁLOGO VSAM

VSAM utiliza un catálogo para mantener la información de todos los volúmenes y todos los ficheros VSAM. Este catálogo es un fichero especial, siendo su organizacion *KSDS*.

Gracias al catálogo VSAM, esta organización presenta una función nueva con las siguientes características:

- Gestión automática de espacio DAAD:

VSAM conoce gracias al catálogo el espacio disponible en los volúmenes. Con ésto se proporciona el espacio libre y el ocupado. Siempre que se vaya a crear un fichero, VSAM selecciona automáticamente un área libre para ese

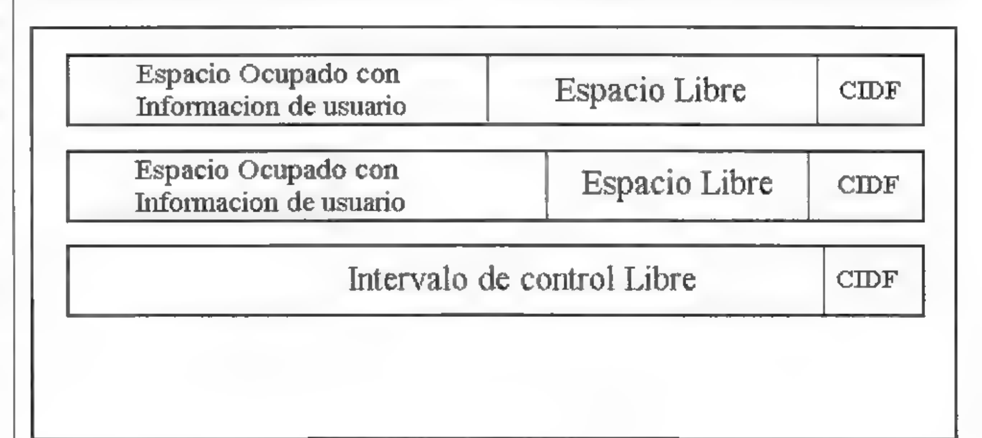


Fig 4.- Intervalos de control libre y campos de control

fichero, y siempre que se desee pedir información sobre un determinado fichero, habrá que proporcionar el nombre de ese fichero al catálogo.
- Protección de datos: VSAM es el que elige el área para un fichero nuevo. No será por tanto posible su destrucción por error, ya que es el propio VSAM el que protege esos ficheros.
- Datos Privados (confidencialidad): Se definen unas palabras claves para poder localizar los datos de un fichero, por lo que, cuando se desee recuperar información sobre un determinado fichero habrá que proporcionar la password o palabra de paso.
- Estadísticas: VSAM mantiene en el catálogo un conjunto de estadísticas de la utilización de las informaciones, al tiempo que almacena en el catálogo toda la información asociada con las características físicas del dispositivo en el que reside la información.

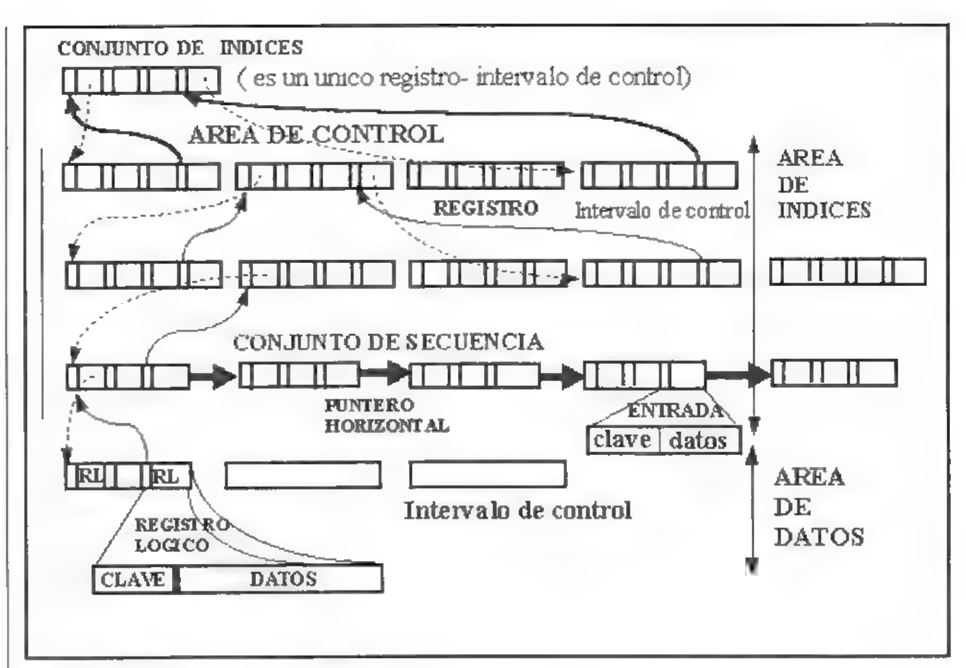


Fig 5.- VSAM: Tipo de fichero KSDS.

AMS (ACCESS MODE SERVICE)

El *AMS* es un conjunto de programas de servicio, o utilidades que permiten:

Un registro de datos, junto con la información de control, es lo que configura el Registro Almacenado

- Definir, imprimir y copiar los ficheros VSAM
- Reorganizar ficheros VSAM
- Añadir, alterar, borrar e imprimir entradas en el catálogo.
- Convertir ficheros *ESDS* en *KSDS*.

CONSIDERACIONES FINALES

Este artículo, como se dice en la presentación, cierra la miniserie dedicada a los ficheros. Todas las organizaciones y métodos vistos tienen su importancia, pues son todos los que existen y, además, todos están vivos. Unos, los mas complejos, son los utilizados por los Gestores de Bases de Datos (SGDBs) ya que al tiempo que gestionan la complejidad establecen una barrera casi infranqueable ante los datos que mantienen (organizacion directa). Otros, más sencillos, secuencial, secuencial Indexados y particionados son los comúnmente usados por los programadores. Por último, la Organizacion VSAM es usada tanto por unos como por otros, citando a modo de ejemplo un Gestor basado en ficheros VSAM, el SGDB: *DB1* de la casa *SAPIENS*.

UTILIDAD

La utilidad que acompaña este mes al artículo, consiste en una *clist* que lista del catálogo de usuario todas las entradas que cuelgan del nivel que se especifique, pasándolas a un fichero para su posterior tratamiento.

La utilidad está contenida en los siguientes ficheros:

-**UTILIDAD.JMP**: *Hardcopys* de tres paneles en tiempo de ejecución de la utilidad.

- El primero presenta un menú de utilidades, desde el que se puede acceder a la utilidad que acompaña al artículo.
- El segundo presenta el panel que aparece al elegir la opción 6 del menú anterior. En él se puede especificar el nivel cualificador de las entradas a listar del catalogo.
- El tercero presenta el resultado obtenido.

- **CATALOGO.CLS**: *CLIST* central de la utilidad. Básicamente, el esquema de este programa de comandos consiste en pedir mediante un panel el nivel cualificador a listar. Ejecuta el comando *LISCAT* de *TSO* con el parámetro *LEVEL* para obtener todas las entradas del catálogo que cuelgan del prefijo especificado y pasar esa salida a un fichero, de forma parecida a como se comentó en los artículos de esta sección correspondientes a los números 11 y 12 de esta revista. Una vez hecho esto, el problema se reduce a tratar el fichero obtenido como una tabla, presentándola mediante un panel.

- **CATALOGO.PAN** : Definición del panel invocado por la *clist* para pedir el nivel cualificador.

- **ENTORNO0.PAN** : Definición del panel utilizado para presentar los resultados.

-**LIBRO.DOC** : El autor de este artículo ha escrito un libro, inédito hasta la fecha, titulado *"TSO para desarrolladores"* en el cual describe los conceptos básicos de MVS y ya más exhaustivamente, con ejemplos concretos, los tres apartados que debería conocer todo desarrollador en una gran instalación *JCL*, *CLIST* e *ISPF*. El fichero citado contiene una presentación de dicho libro.



SHADING BOBS

Pedro Antón

En este artículo se explicará cómo crear este antiguo efecto, pero además se aprovechará para realizar una introducción en el mundo del coprocesador matemático

Éste es otro de esos efectos sencillos que se consiguen a partir de un tratamiento adecuado de la paleta y un sencillo *sprite*. La vistosidad de este efecto radica principalmente en dos aspectos. El primero de ellos es la elección de la paleta de colores, mientras que el segundo consiste en las diferentes formas que pueden adoptar las curvas.

EL EFECTO SHADING BOBS

Al igual que en el "blur motion", en este efecto también se mantiene una sombra o estela (figura 1).

Este efecto toma su nombre de la estela que dejan las bolas (o *bobs*) al moverse por la pantalla. Su diseño es muy sencillo, y se basa en la variación del color del *sprite* al ponerlo en la pantalla o buffer cuando ya hay un *sprite* o cualquier otro gráfico debajo de éste. Como intuirá el lector, únicamente se debe aplicar algún tipo de operación entre el color del *sprite* y el existente. En este caso, una suma, pero debe quedar claro que, como se mencionó hace un par de artículos, experimentando con distintos efectos, se pueden conseguir efectos muy llamativos, y algunas veces, nuevos, algo muy importante en el mundo de las demos.

A continuación, se entrará de lleno en el análisis del código que acompaña a este artículo. En primer lugar, se puede observar que la creación de la paleta adecuada es imprescindible en el efecto. La paleta será creada por el programa y consiste en dos degradados consecutivos de negro a azul y de azul a blanco, aunque insistiendo en que el lector podrá modificarla a su gusto. Los degradados creados usan 64 colores cada uno, sumando un total de 128 colores, que es la mitad de la paleta. Por ello, debe limitarse en la rutina para que ponga los *sprites* el límite de color a 128.

Esto se realiza con una simple operación lógica del color total a poner con 127 (listado 1). Por lo tanto los pasos a seguir a la hora de poner el *sprite* serán:

1- Calcular la dirección donde se va a colocar el *sprite*.
2- Leer el byte a poner.
3- Comprobar si es cero, en cuyo caso no se pondrá nada.
4- Leer el valor del byte almacenado en la pantalla o en el buffer.
5- Sumarle a dicha cantidad el valor del dato a poner.
6- Realizar con dicho dato la operación lógica *AND* con 127, que es el número de colores de la paleta que se ha generado.

Y esto es todo lo que hay que hacer para conseguir este efecto. Sencillo ¿verdad?

LAS CURVAS

Las curvas empleadas en el ejemplo son las famosas curvas de Lissajous.

Es evidente que las curvas que se deseen usar con este efecto no han de ser curvas simples. Ésta será la única forma de que el efecto quede un tanto espectacular. El ejemplo que acompaña a este articulo emplea curvas de Lissajous. Éstas consisten en la composición de movimientos armónicos simples de direcciones perpendiculares, o dicho de otra forma, los valores de x vienen dados por una señal armónica simple, y los de y por otra distinta.

Un movimiento armónico simple viene definido por tres variables: la amplitud, la frecuencia y el desfase de la señal (ver figura 2).

Como las curvas de Lissajous, son la composición de dos, la figura creada viene definida por la relación entre las variables:

- Las amplitudes darán mayor o menor simetría a la señal.
- La relación de frecuencias dará en número de oscilaciones de la composición por período. Como las señales empleadas en el ejemplo son las mostradas en la figura 3, el período de una oscilación completa será dos *pi*.
- El desfase entre ambas señales.

Es fácilmente imaginable que los valores que van a tomar las curvas no van a ser sencillos de calcular. Normalmente, esto se soluciona usando tablas con una buena exactitud, es decir, usando más de un byte para almacenar cada valor de seno o coseno. Sin embargo, en este caso, se va a aprovechar el ejemplo para comentar cómo funciona el coprocesador matemático.

EL COPROCESADOR MATEMÁTICO

La familia de coprocesadores 80x87 fue creada para agilizar los cálculos matemáticos cuando se trabaja con números reales, BCD, o enteros largos.

Es raro el juego o la demo que se crea hoy que no requiera el uso de una máquina potente, es decir, un 486 o superior. Es sabido por todos que estos procesadores incluyen en el mismo, encapsulado, procesador y coprocesador. Por lo tanto, ¿por qué no emplearlo?. A continuación se comentará de forma sencilla cómo comenzar a trabajar con el coprocesador.

Los pasos a seguir cuando se desee trabajar con el coprocesador son los siguientes:

Este efecto toma su nombre de la estela que van dejando las bolas o Bobs al moverse por toda la pantalla

1- Cargar los datos de la memoria a los registros del coprocesador.
2- Procesar los datos.
3- Almacenar los datos procesados en la memoria.

Los pasos 1 y 3 deben realizarse de forma sincronizada con el procesador. Sin embargo, el paso 2 puede ser realizado mientras el procesador realiza otras tareas.

Para analizar el funcionamiento del coprocesador, es necesario saber que éste posee ocho registros de datos de 8 bits. La sintaxis para especificar uno de estos registros es:

ST (Numero de registro)

Si se omite el número de registro, el coprocesador asumirá una referencia al registro 0.

Existen varios modos de trabajo,

LISTADO 1

```
;
; PUT PALETTE
;
PutPal  PROC
        PUSH   ES DI AX CX DX
        CLI
        MOV    DX,03C8h
        XOR    AL,AL
        OUT    DX,AL
        ST.
        MOV    AX,DS
        MOV    ES,AX
        XOR    CX,CX
@@RGBLoop1:
        CLI
        MOV    AL,CL
        OUT    DX,AL
        INC    DX
        MOV    AL,0
        OUT    DX,AL
        INC    DI
        MOV    AL,0
        OUT    DX,AL
        INC    DI
        MOV    AL,CL
        OUT    DX,AL
        INC    DI
        STI
        DEC    DX
        INC    CL
        CMP    CL,64
        JB     @@RGBLoop1
        MOV    CH,CL
@@RGBLoop2:
        CLI
        MOV    AL,CL
        OUT    DX,AL
        INC    DX
        MOV    AL,CH
        OUT    DX,AL
        INC    DI
        MOV    AL,CH
        OUT    DX,AL
        INC    DI
        MOV    AL,63
        OUT    DX,AL
        INC    DI
        STI
        DEC    DX
        DEC    CH
        INC    CL
        CMP    CL,128
        JB     @@RGBLoop2
        POP    DX CX AX DI ES
        RET
PutPal  ENDP
```

Generación de la paleta.



Ejemplo de efecto Shading Bobs.

LISTADO 2

```
@@MainLoop:
        MOV    t,0
 @@OnePeriod:
 ; x=160+(A1*Sin(t))
        FINIT          ; Resets the coprocesor
        FLD    t       ; Load t
        FCOS           ; ST = Cos (t)
        FIMUL  A1      ; ST=ST*A1
        FIADD  CenterX ; ST=ST+CenterX
        FIST   Temp
        MOV    AX,[Temp]
        PUSH   AX
 ; y=90+(A2*Cos((Incw*t)+f))
        FINIT
        FLD    t       ; Load t
        FIMUL  Incw    ; ST=ST*Incw
        FADD   f       ; ST=ST+f
        FCOS           ; ST = Sin (t)
        FIMUL  A2      ; ST=ST*A2
        FIADD  CenterY ; ST=ST+CenterY
        FIST   Temp
        MOV    DI,[Temp] ; DI=ST

        MOV    DX,03DAh  ; Wait for Vertical
@@Jump1:                 ; retrace ;)
        IN     AL,DX
        TEST   AL,08h
        JZ     @@Jump1
@@Jump2:
        IN     AL,DX
        TEST   AL,08h
        JNZ    @@Jump2

        POP    AX
        PUSH   SI
        MOV    SI, OFFSET Sprite
        MOV    BL,16
        MOV    BH,BL
        CALL   PutSprSh
        POP    SI

        CMP    CS:[EscKey],0
        JNE    @@GoOUT
 ; t=t+Inct
        FINIT
        FLD    t
        FADD   Inct
        FST    t
       FCOMP  DosPI    ; Compares ST to DosPI
        FSTSW  AX       ; Store control word in
AX
        SAHF
        JC     @@OnePeriod; Jump if t<DosPI
 ; f=f+Incf
        FINIT
        FLD    f
        FADD   Incf
        FST    f
       FCOMP  DosPI    ; Compares ST to DosPI
        FSTSW  AX       ; Store control word in
AX
        SAHF
        JC     @@TestKey
        MOV    f,0
@@TestKey:
        CMP    CS:[EscKey],0
        JE     @@MainLoop
@@GoOut:
```

Cálculo de las curvas de Lissajous.

pero los cuatro más simples son los siguientes:

- Operación inmediata del registro.
- Operación entre los dos primeros registros *ST(0)* y *ST(1)*
- Operación entre registros específicos.
- Operación entre memoria y *ST(0)*

$$x(t)=A\cos(wt+\phi)$$

Donde: A- Amplitud

w- Pulsación $= \frac{1}{f}$

ϕ- Desfase

Definición de movimiento armónico.

Además de estos cuatro modos, existe otro más, pero no se estudiará en este artículo.

El primer paso y el último es cargar los datos de la memoria al coprocesador, y almacenar los datos del coprocesador en la memoria. Estas operaciones se realizan principalmente mediante las instrucciones *FLD* y *FST* y admiten, como es de suponer, todos los modos de trabajo. La primera de ellas, *FLD*, carga los datos a un registro del coprocesador. La segunda, *FST*, almacena los datos del coprocesador. Principalmente se usan de la forma:

La utilización del coprocesador dará una mayor velocidad a los cálculos que hay que realizar en la demo

FLD/FST mem: Copia el dato de la dirección de memoria especificada en el registro *ST(0)* o almacena el registro *ST(0)* en la memoria.
FLD ST(num): Almacena en *ST(0)* el dato del registro *ST(num)*.
FST ST(num): Almacena en *ST(num)* el dato del registro *ST(0)*.
FXCH ST(num): Intercambia el dato de los registros *ST(0)* y *ST(num)*. Si no se especifica nada, se intercambian los datos de *ST(0)* y *ST(1)*.

Las instrucciones de carga más generales son:

FLDZ: Almacena un cero en *ST(0)*.
FLD1: Almacena un uno en *ST(0)*.
FLDPI: Almacena en *ST(0)* la constante *PI*.

A continuación se verán algunas de las operaciones aritméticas que realiza el coprocesador:

FADD: Suma *ST(0)* y *ST(1)* almacenando el resultado en *ST(0)*.
FADD mem: Suma *ST(0)* y el dato almacenado en la dirección de memoria *mem*. El resultado se almacena en *ST(0)*
FIADD mem: Igual que *FADD*, pero el dato de la memoria debe ser entero.
FSUB: Resta a *ST(0)* *ST(1)* y almacena el resultado en *ST(0)*.
FSUB mem: Resta a *ST(0)* el dato que contenga la dirección de memoria *mem*. Almacena el resultado en *ST(0)*
FISUB mem: Igual que *FSUB*, pero el dato a restar debe ser un número entero.

Además de estas operaciones de suma y resta, existen otras que operan entre registros. Por ejemplo, restar al valor de *ST(5)* *ST(0)* y almacenar el resultado en *ST(5)*. También existen operaciones de resta invertida, es decir, restan al destino el origen, en lugar de al revés.

A continuación vienen las operaciones de multiplicación y división:

FMUL: Multiplica *ST(0)* y *ST(1)* y almacena el resultado en *ST(0)*.
FMUL mem: Multiplica *ST(0)* y el dato

$$x(t) = A \cos(w_1 t)$$

$$y(t) = A \cos(w_2 t + \phi)$$

Formulas de las figuras de Lissajous.

almacenado en la dirección de memoria *mem*. Como es de suponer, almacena el resultado en *ST(0)*.
FIMUL mem: Realiza la misma operación que *FMUL*, pero el dato que se encuentra en la dirección de memoria *mem* deberá de ser un entero.
FDIV: Divide *ST(1)* entre *ST(0)*, almacenando el resultado en *ST(0)*.
FDIV mem: Divide *ST(0)* entre el dato contenido en la dirección de memoria *mem*.

El proceso de datos en el coprocesador puede realizarse mientras el procesador hace otras taréas

FIDIV mem: Igual que *FDIV* pero el dato divisor deberá ser entero.

Al igual que ocurre con la suma y la resta, también con éstas operaciones aritméticas existen instrucciones que dividen y multiplican registros. Además existen las divisiones invertidas, como ocurre con las restas.

Otras instrucciones interesantes son:

FABS: Modifica el valor de *ST(0)* dejándolo siempre positivo (valor absoluto).
FCHS: Cambia el signo de *ST(0)*.
FRNDINT: Redondea *ST(0)* a un entero.
FSQRT: Almacena en *ST(0)* el valor de su raíz cuadrada.
FSIN: Calcula el valor del seno de *ST(0)* en radianes y lo almacena en el mismo registro.
FCOS: Almacena en *ST(0)* el valor del coseno de *ST(0)*.
FSINCOS: Calcula el seno y el coseno de *ST(0)*. En *ST(0)* se almacena el coseno y en *ST(1)* el seno.

Como es de suponer, es interesante saber si el resultado de una o varias operaciones es cero, mayor o menor, etc, es decir, controlar el resultado de las operaciones, para que el código creado pueda obrar en consecuencia. Para ello existen, entre otras, las siguientes instrucciones:

FCOM: Compara *ST(0)* y *ST(1)*.
FCOM ST(num): Compara *ST(0)* y *ST(num)*.
FCOM mem: Compara *ST* con el dato almacenado en la dirección de memoria *mem*. Debe recordarse que, si se omite el registro al referenciar *ST*, se asume referencia al registro *ST(0)*.
FICOM mem: Igual que *FCOM* pero el operando en memoria debe ser entero.
FTST: Compara *ST* con cero.

Ahora bien, si, como se ha comentado al principio de este artículo, el procesador y el coprocesador son independientes, ¿cómo afectará el resultado de una operación al control del programa?. Es obligado, ahora, hablar de las banderas de control del coprocesador. El coprocesador posee cuatro banderas cuyo contenido depende del resultado de las operaciones del coprocesador (ver figura 4).

Estas banderas se corresponden, en posición, con algunas del registro de estado del procesador. En concreto, se corresponden las banderas de *cero*, *paridad* y *carry*, como se puede observar en la figura 5.

El problema es transferir el control al procesador con esas banderas de estado. Esto debe realizarse de la siguiente forma:

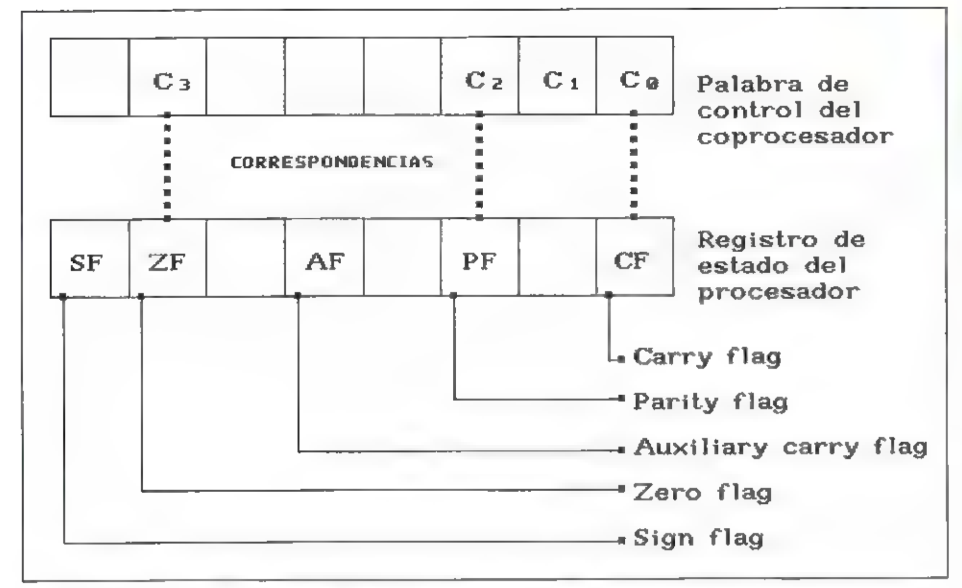
- Realizar la comparación.
- Almacenar la palabra de control en la memoria, o en *AX*.
- Escribir en el registro de estado la palabra de control.

Escribir esto en ensamblador es sencillo, como se verá más adelante, pero para ello se debe comentar la instrucción *FSTSW AX*. Esta instrucción almacena la palabra de control en *AX*.

Para terminar con esta introducción sobre el uso del coprocesador, la instrucción *FINIT* inicializa el coprocesador. Esta instrucción es muy útil para así evitar que operaciones realizadas anteriormente influyan en los resultados en curso.

Despues de FCOM	C_3	C_2	C_1
ST>Origen	0	0	0
ST<Origen	0	0	1
ST=Origen	1	0	0
No comparables	1	1	1

Resumen de estado de las banderas.



Banderas de estado del coprocesador.

LAS CURVAS DE LISSAJOUS Y EL COPROCESADOR

A la hora de implementar este efecto dentro de una demo, el uso del coprocesador, agilizará los cálculos enormemente.

Como el lector podrá observar, en el código que acompaña a este artículo no se calculan los senos y cosenos mediante el uso de tablas, sino que se implementa el uso del coprocesador con la finalidad de ilustrar con un ejemplo lo explicado.

comprobar en cualquier otro programa si el redondeo es posible (no afecta al rsultado final), o si por el contrario debe utilizar cálculos exactos.

Otro punto a destacar en el listado es la comparación de las variables *t* y *f* con dos *pi*. En la comparación realizada en el bucle interior se usa el salto condicional *JC* (*Jump if carry*), ya que el programa saltará a la etiqueta *@@OnePeriod* siempre y cuando la variable *t* es menor de dos *pi* y, una vez transferida la palabra de con-

A diferencia de las operaciones internas del coprocesador, el intercambio de datos con el procesador requiere sincronización

En el listado 2 se puede observar el uso de la instrucción *FINIT* cada vez que se va a hacer uso del coprocesador para realizar un nuevo calculo. Esto asegura la no interferencia entre distintos resultados de operaciones independientes. Además, es de destacar el uso de las instrucciones del coprocesador con números enteros. Ello es debido a que la precisión necesitada por el código del ejemplo no es excesiva y se puede realizar un redondeo a enteros a partir de la multiplicación del coseno por la amplitud de la onda. Este redondeo no siempre será posible, por lo que lector debrá

trol del coprocesador al registro de estado del procesador,el banderín de *carry* estará activo si dicha comparación es menor. Como el lector podrá observar, algo análogo ocurre con el bucle principal.

CONCLUSIÓN

Esto es todo por esta vez. Es casi seguro que el uso del coprocesador servirá para hacer más rápido el código de los lectores. Como siempre, se anima a los lectores a realizar sus propias prácticas con el fin de familiarizarse con el uso del coprocesador en sus propios programas.



EMULADORES DE MS-DOS

David Aparicio

MS-DOS cuenta con más aplicaciones nativas que ningún otro sistema para ordenador personal que haya existido hasta ahora. Este mes se verá los esfuerzos del mundo Linux por aprovechar estos programas

DOS es un sistema operativo de 16 bits, monotarea, simple y obsoleto. A pesar de todo ello, ha representado, por más de una década, lo que ningún otro sistema: un estándar sobre el que desarrollar un abanico innumerable de aplicaciones. Como resultado, los programas disponibles representan un parque nada desdeñable, que debe ser reaprovechado de alguna forma. Además, el recubrimiento gráfico por excelencia para DOS, Windows, ha añadido programas todavía más vistosos, junto a una necesaria estandarización de programación.

Cualquier sistema operativo que pretenda tomar el relevo en los ordenadores personales actuales, debe ofrecer al usuario una compatibilidad hacia atrás que permita seguir amortizando las aplicaciones antiguas, y con las que los usuarios se sienten a gusto. Las personas que colaboran en el desarrollo de Linux también están interesadas en unificar todas las aplicaciones bajo un sólo sistema operativo.

Para los programas MS-DOS aparece un emulador, "*dosemu*", que proporciona una máquina virtual sobre la que ejecutar los programas. Para Windows "*wine*" representa un traductor de llamadas del entorno de MicroSoft al de Linux. Antes de todo, se ha de comentar que, mientras que la emulación DOS se encuentra en un estado muy avanzado, como se verá a lo largo de este artículo, el segundo emulador es todavía un "juguete" que sólo sirve para los usuarios más curiosos.

NOCIONES DE HARDWARE

Para entender mejor el método de emulación que emplean estos programas, debe observarse primero las capacidades disponibles en el hardware de un PC actual. Mientras que un 8086 no dispone de capacidades multitarea, y es incapaz de direccionar segmentos de más de 64 Kbytes, y un 286 tiene una capacidad multitarea muy limitada, la familia 386 y modelos superiores tienen un diseño especializado. Hay que recordar que Linux necesita un microprocesador 386SX o superior para funcionar.

En primer lugar, el direccionamiento con registros de 32 bits posibilita la creación de aplicaciones donde el tamaño de las estructuras no estén limitadas a los molestos segmentos de 64 Kbytes. Adicionalmente, la *MMU* permite la implementación de memoria virtual (capacidad para utilizar parte del disco como una extensión de la memoria RAM), asignación lógica de direcciones (las direcciones reales de RAM se pueden *mapear* como otras diferentes sin hacer movimientos físicos de datos), la protección entre programas (un proceso no puede, por errores propios, destruir zonas de memoria pertenecientes a otro proceso), y los niveles de privilegio (el núcleo central ejecuta a máxima prioridad, mientras que otros procesos pueden distribuirse en hasta tres capas adicionales, con diferente nivel de privilegio).

Además, la arquitectura 386 permite implementar máquinas virtuales en un sólo procesador. Dado que puede separarse la memoria en varias zonas numeradas de forma independiente, que hay varios niveles de privilegio, y que todos los accesos a puertos externos pueden ser supervisados y redirigidos por el núcleo central, un 386 es capaz de ejecutar varias sesiones 286 y 8086 por hardware, y situarlas en cual-

quier parte de la memoria, sin mayor límite que el que su propia potencia imponga.

El emulador de DOS crea una sesión en una máquina virtual aislada (híbrida entre un 286 y un 386 real), y se encarga de simular los accesos a memoria de vídeo y a puertos de entrada/salida para hacer creer a las aplicaciones que se ejecutan dentro de la máquina virtual que disponen de un hardware de PC completo. En la idea original, el emulador construía una reproducción de un PC, y dejaba que una copia real de MS-DOS se ejecutase íntegramente en él. Actualmente, cada vez más *drivers* DOS se integran en la emulación, (*XMS*, *EMS*, terminal ANSI, ratón y teclado) para conseguir dos cosas: mayor rapidez de ejecución y más memoria principal libre. Si se observa la figura 1, es posible ver unos 620 Kbytes libres en una sesión que incluye *IPX* (red Novell), CD-ROM, EMS y ratón, lo que, paradójicamente, no es posible con dicho sistema operativo en la realidad.

Por otra parte, *wine* ("*WIne is Not Emulation*") toma una aproximación diferente. Dado que las aplicaciones para Windows tienen un estándar de llamadas (*API*) definido, y no acceden al hardware de forma directa, la creación de una máquina hardware en donde correr Windows no es necesaria. Con X-Windows ejecutando sobre Linux, se tiene un superconjunto de las llamadas que el entorno de Windows proporciona, y la labor del emulador consiste en hacer una traducción en tiempo real de las llamadas a funciones, y las respuestas necesarias para la aplicación.

Esta solución permite una mayor integración con el resto del sistema (en la figura 2 las aplicaciones Windows se mezclan con las nativas), aunque es más complicada de implementar, debido principalmente a muchas funciones no documentadas que sólo utilizan las propias aplicaciones de MicroSoft. Aunque esto representa, cuantitativamente, a un reducido parque de programas, la realidad es que éstos no pueden ser ignorados: Word 6.0, Excel, Access y otros representan un gran peso específico de la ofimática habitual. Adicionalmente, las inconsistencias

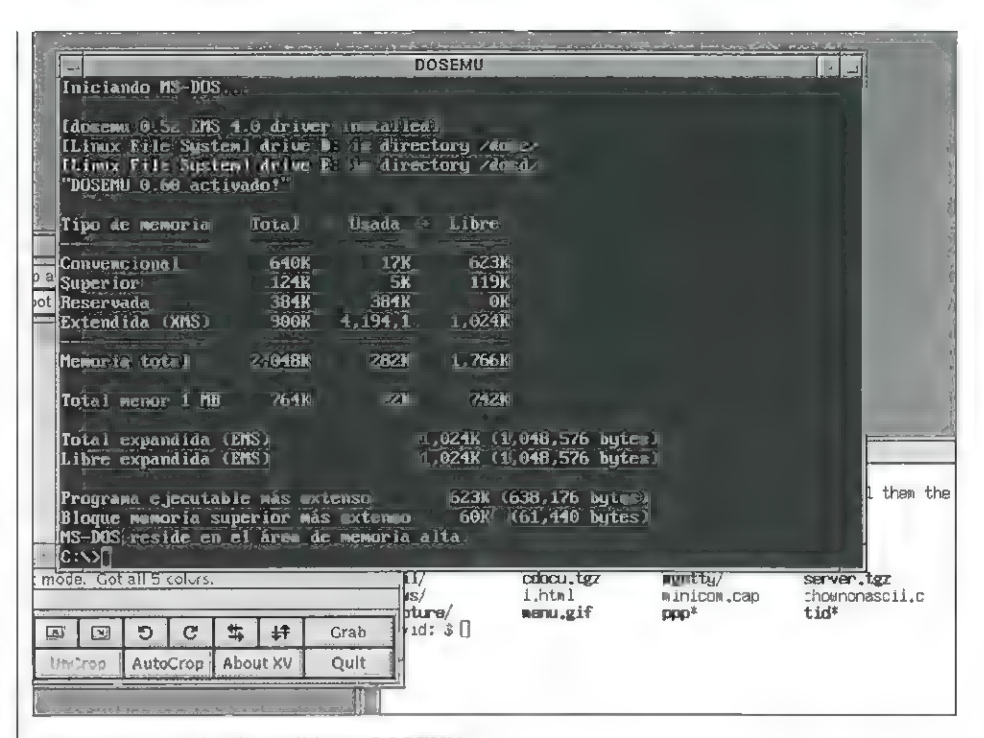


Figura 1. Memoria disponible en DOSEMU.

internas que se están produciendo con la migración a 32 bits, hacen que mantener compatibilidad en la emulación no sea tarea sencilla.

DOSEMU: APURANDO EL HARDWARE

Como ya se ha mencionado, el éxito de este emulador se basa en conseguir la mejor aproximación posible al comportamiento del hardware de un PC. Por supuesto, con ningún procesador se puede conseguir una emulación rápida de sí mismo. Por tanto, la ejecución de una aplicación DOS en una sesión de emulación con *DOSEMU* varía entre un 90% del rendimiento del sistema real, para programas de texto, a un 50% para aplicaciones gráficas complejas. En la práctica, la pérdida de velocidad es aceptable, si a cambio se consegue ejecutar varias cosas simultáneamente.

El emulador tiene dos aspectos separados, puesto que se puede ejecutar tanto desde una consola (es decir, con el teclado y a pantalla completa, en el PC que corre Linux, como hace DOS en la realidad) como desde un terminal o dentro de X-Windows (pantalla y teclado remotos, en ambos casos).

En el primer caso, el emulador se invoca simplemente con:

dos

mientras que para X-WIndows, se invoca lo siguiente desde una ventana:

xdos

A diferencia de las sesiones de texto, que funcionan igual de bien en cualquiera de los tres modos (vea un ejemplo del conocido Turbo Pascal 6.0 dentro de X-Windows en la figura 3), la ejecución con gráficos requiere acceso desde la consola local, si se desea una velocidad aceptable.

En cuanto al sistema de almacenamiento, es posible usar tanto una partición con MS-DOS, como un fichero con un formato interno especial (disco imagen), o bien el propio sistema de ficheros de Linux.

El acceso a red, puertos serie, impresora y discos flexibles se realiza con un interfaz que comunica a los *drivers* del propio núcleo de Linux con la sesión de emulación.

La ejecución de una sesión de DOS puede ser lanzada por los usuarios que hayan sido autorizados, mediante un fichero de configuración, */etc/dosemu.users*, e incluso se pueden lanzar varias sesiones simultáneas, siempre que haya restricciones en el acceso a las unidades de almacenamiento comunes.

En el disco que acompaña a la revista, se ha incluido todo lo necesario para instalar el emulador. Una vez copiado y descomprimido el archivo *dosemu.tgz*,

se obtendrán los siguientes componentes:

• *hdimage*: Un fichero que contiene una imagen de arranque (C:) para el emulador, ya preconfigurada.

• *dosemu.conf*: El fichero de configuración del emulador.

• otros ficheros informativos (en inglés) y utilidades.

Con el fichero de configuración y la imagen de arranque, se dispone de un entorno listo para empezar sin problemas una sesión de emulación. Necesitará tener instalado previamente el emulador de DOS que se suministró, por ejemplo, en el CD del número 18 (*SlackWare 3.00 ELF*) en el directorio */linux300/contrib/dosemu60.tgz*.

A continuación, se examinará el fichero de configuración, que se divide en varios apartados. Observesé la tabla 1 para seguir el ejemplo. Las tablas se encuentran en el CD que acompaña la revista.

1 - Recursos de la sesión de emulación (DEBUG, MISCELLANEOUS, BOOT).

Aquí se indican los recursos que el emulador debe proporcionar a la sesión DOS, y las auditorías que deben realizarse sobre la misma. El apartado de depuración mostrará, por el terminal desde el que se ha lanzado *dosemu*, mensajes sobre las operaciones que se están realizando, y los accesos ilegales al hardware. Por ejemplo, si se protege un disco contra escritura, y un programa intenta este tipo de acceso, se verán mensajes al respecto en el terminal adecuado.

También se indica si se desea una pantalla de bienvenida del emulador (*dosbanner*) y si se desea una interrupción para que DOS disponga de un *timer* preciso (*timint*), el tipo de procesador (*cpu* y *matco*), y desde qué unidad se arrancará (*bootC* ó *bootA*). Este último parámetro es imprescindible para configurar inicialmente el emulador, puesto que en las primeras ocasiones se debe comenzar con un disco flexible, configurando la imagen de disco duro desde la que se arrancará normalmente.

Para el arranque desde un disco duro que también se utilice para iniciar el DOS real, se dispone de la posibilidad de tener un *config.sys* y *autoexec.bat* para la emulación con diferente extensión, y facilitando así la diferenciación de configuraciones. Esto se controla con los parámetros *EmuSys* y *EmuBat*. Por cierto, que si ésta es la configuración que tenía pensada, y tiene *LILO* en el *MBR*, debe saber que el emulador no es capaz de arrancar de esta unidad.

2 - Unidades de almacenamiento (HARD DISK, FLOPPY DISK)

Hay dos formas de disponer de unidades de almacenamiento físicas en el emulador: bien por un dispositivo especial, del directorio */dev* (tanto particiones como dispositivos enteros, *IDE* o *SCSI*), o por un fichero Linux que contenga internamente una estructura de unidad DOS, denominado por ello "imagen de disco duro". Este aspecto se controla con el parámetro *disk*. Nótese que el orden en que se definen dispositivos o ficheros en la configuración es relevante para la asignación de unidades DOS. La primera referencia de "disk" será la unidad C:, y así sucesivamente.

En el disco de la revista se distribuye una imagen (*/var/lib/dosemu/hdimage*) ya preparada. Si se desea crear un disco propio, de tamaño 'N', se deberán realizar las siguientes operaciones:

1. Ir a la línea de comando de Linux
2. Teclear "*cd /var/lib/dosemu*"
3. Teclear "*dd if=/dev/zero of=hdimageC bs=1k count=N*"
4. Teclear "*dd if=/etc/dosemu/sign of=hdimageC*"
5. Añadir "*hdimageC*" como una unidad más, en */etc/dosemu.conf*
6. Seleccionar opción "*bootA*" en */etc/dosemu.conf*
7. Introducir un disco de instalación DOS en A:
8. Arrancar *DOSEMU*
9. Particionar y formatear la unidad con DOS
10. Copiar los ejecutables
11. Si va a ser una unidad de arranque, y la imagen está como unidad C:, usar el comando "*FDISK /MBR*"
12. Seleccionar opción "*bootC*" en */etc/dosemu-.conf*

Las unidades de disco flexible se definen de forma similar a las fijas, con los parámetros *floppy* y *bootdisk*, para tener unidades reales o simuladas de un fichero imagen, respectivamente. El fichero imagen se construye a partir de



Figura 2. Convivencia de programas Windows y Linux

un disco real, con el comando:

dd if=/dev/fd0 of=/path/imagen bs=16k

Repase con "*man dd*" este comando, que se asemeja a un "diskcopy" de DOS, sólo que más versátil, permitiendo usar ficheros o unidades indistintamente, copiando todo o parte de la unidad, y con tamaños de bloque definibles.

3 - Emulación en la consola (KEYBOARD, VIDEO).

Cuando se ejecuta el emulador a pantalla completa en un PC, debe indicarse ciertos aspectos de la tarjeta de vídeo, así como el comportamiento del teclado en la emulación. En la tabla adjunta, el comando *keyboard* y *video* controlan dicho comportamiento.

Hay que seleccionar, según el ejemplo, el teclado español (por tanto no se debe ejecutar el comando *KEYB* dentro del DOS, ya que se desconfigurará el teclado preparado por la emulación), con interrupciones de teclado (*keybint*) y modo trasparente (*rawkeyboard*). Estos dos últimos parámetros estarán activos si se desea una emulación perfecta donde ejecutar la mayor parte de programas. Como contrapartida, mientras se tengan estos parámetros activos es arriesgado cambiar de consola (entre sesiones DOS y Linux) cuando corre la emulación, puesto que es posible perder el control de teclado (en realidad Linux sigue ejecutando, pero no se pueden introducir comandos, y sólo queda el recurso del botón de reset, excepto que se disponga de conexiones por red o línea serie).

En cuanto al vídeo, se tiene emulación en modo sólo texto, así como de modo gráfico (con *graphics*). Algunos

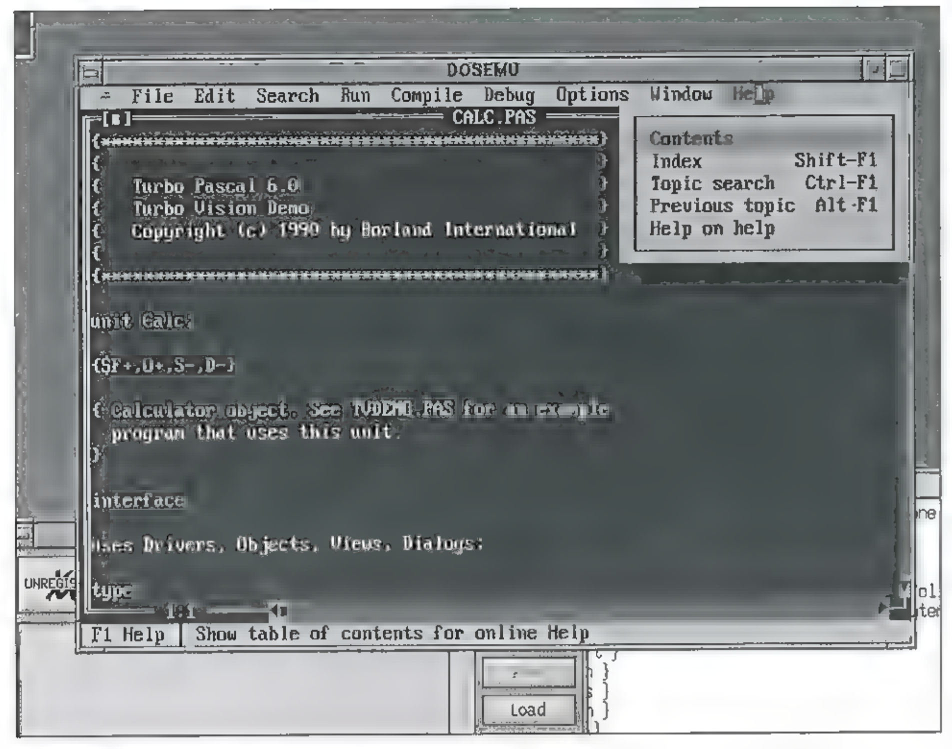


Figura 3. Turbo Pascal dentro de una emulación DOS.

modelos de tarjetas pueden no funcionar correctamente en este último, necesitando parámetros adicionales. Algunas "recetas" útiles si el vídeo no arranca correctamente son:

- Deshabilitar memoria "*shadow*" o "*caché*" para la BIOS de vídeo.
- Poner la opción "*allowvideoportaccess on*" en el fichero de configuración.
- Utilizar la opción "*ports*" (ver más adelante) para permitir acceso a puertos especiales de vídeo.
- Especificar el parámetro "*vbios_seg XXXX*" para indicar la dirección del BIOS de vídeo en modo real.
- Copiar la BIOS de vídeo a un fichero imagen, (con el comando "*getrom >vgarom*") y arrancar el emulador con la opción de vídeo *vbios_file "/path/vgarom"*.

Aunque el modo consola permite ejecutar muchas aplicaciones DOS correctamente, el comportamiento del emulador ante un imprevisto suele ser más catastrófico. Habrá que practicar un poco con la configuración antes de confiar en *DOSEMU* para realizar algún trabajo trascendente.

4 - Emulación en terminal y X-Windows (TERMINALS, X-SUPPORT)

La emulación por terminal no es apropiada para modos gráficos, pero proporciona un uso más cómodo de programas, tales como los entornos de compilación de Borland, editores sencillos, y aplicaciones de texto en general. El comportamiento del emulador es muy estable y fiable en este modo. Quizá en un futuro los desarrolladores incorporen capacidades gráficas limitadas.

5 - Líneas serie, ratón, red e impresoras (SERIAL, NETWORKING, PRINTERS)

Es posible controlar el acceso a puertos serie, paralelos y de red (con una emulación *Novell IPX*), a través de los dispositivos reales (caso de puertos serie para ratón, o puertos paralelo, para impresoras), o bien de filtros intermedios (impresión de documentos por red *TCP/IP* o en ficheros). En el ejemplo, pueden observarse varias líneas de configuración, con las opciones "serial", "mouse" y "printer".

6- Memoria, puertos hardware e interrupciones (MEMORY, IRQ, PORT ACCESS)

La ejecución del emulador, como ya hemos comentado, se basa en la creación de una máquina virtual donde hacer creer a una copia del DOS que dispone de un PC completo. En este último apartado se define cuánta memoria se otorga a la sesión de emulación (*dosmem*), de qué tipo es el acceso a memoria por encima del primer mega (*dpmi*, *xms* y *ems*), así como acceso a puertos de entrada salida.

Este último apartado es el más complejo de ajustar, puesto que requiere varias aproximaciones por prueba y error. En el emulador, está previsto el acceso a puertos comunes, como vídeo VGA, puertos serie, programación de DMA, y otros. Además, el soporte de ciertos extensores de DOS posibilita la compatibilidad inmediata con muchos juegos que se basan en estos entornos.

Sin embargo, todavía quedan puertos de E/S que son utilizados de forma imprevisible. Se necesita algún conocimiento de ensamblador para ver los puertos que son accedidos por un programa que no ejecuta "a la primera" en el emulador, y habilitarlos en la configuración. En un ejemplo proporcionado por los autores del emulador, el juego "SimEarth" necesita la siguiente línea:

ports { 0x388 0x389 }

7- Sonido

El soporte de sonido en el emulador es aún limitado. La opción "*speaker*" permite o inhibe el acceso al altavoz del PC. Por ejemplo, si se ejecuta el emulador en red, no tiene sentido "castigar" a la persona sentada frente a la consola con sonidos procedentes de un programa ejecutado en otro terminal.

El soporte de tarjetas de sonido, en fase inicial, tiene como objetivo proporcionar una SoundBlaster Pro a partir cualquier otra tarjeta soportada por el *driver VoxWare* de Linux. Por el momento, sólo está implementado el soporte de sonido DMA en reproducción a 8 bits.

Si se van a ejecutar juegos complejos dentro del emulador, (entre los compatibles ya se encuentran títulos como "Descent", "Dark Forces", "Mortal Kombat 2" o "Rise of the Triad"), hay que recordar que debe deshabilitarse el sonido.

8 - Drivers dentro de la emulación.

Como se ha introducido en los puntos anteriores, algunos *drivers* tradicionales del DOS no deben ser ejecutados dentro de la emulación, que dispone de sus propios substitutivos. Esto afecta, a los manejadores de memoria, ratón, teclado y cdrom. En la tabla 2 se presenta un ejemplo de la configuración del DOS para su ejecución dentro del emulador. Obsérvese aquí todo el soporte necesario, incluyendo el acceso a parte de la

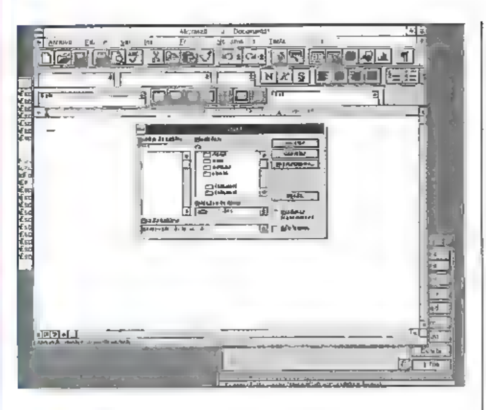


Figura 4. El prometedor futuro del emulador de Windows.

estructura de directorios de Linux, como si se tratase de una unidad de red para DOS.

Por último debe comentarse que, aunque *WINE* intenta la emulación de ejecutables de Windows proporcionando un entorno desde el propio Linux, en el emulador de DOS también se investiga en la ejecución de Windows como tal. Para ello, se usan los ejecutables de Windows que OS/2 proporciona, y los cuales funcionan también dentro de la emulación con *DOSEMU*. Esto último es todavía muy inestable, y los usuarios interesados deberán leer la documentación al respecto que acompaña al emulador.

WINE: TODAVÍA UNA PROMESA

El soporte para programas Windows, mucho más útil para usos profesionales de Linux, se encuentra todavía en una etapa primitiva. La base de aplicaciones que ejecutan bajo este entorno es muy reducida, y son abundantes las posibilidades de caídas durante la ejecución (por supuesto, afectando sólo al emulador, no al resto de Linux). Sin embargo, el progreso realizado ha sido asombroso. En poco más de un año, ya es posible al menos arrancar cosas como Word 6.0 (ver figura 4).

Mientras que el emulador DOS necesita sus propias unidades para poder arrancar y almacenar el sistema operativo, *WINE* aprovecha el sistema de ficheros de Linux para su ejecución. Esto hace más sencilla la instalación. Para ello, serán necesarios los siguientes componentes, disponibles en el fichero *wine.tgz* del disco adjunto:

- El ejecutable, *wine*, que corresponde a la versión del 2 de Marzo del 96.
- El fichero de configuración, *wine.conf*
- Los símbolos de depuración, *wine.sym*
- La página de manual, *wine.1*.

La instalación se consigue, como siempre, con los siguientes comandos:

cd /

tar xvfz /path/wine.tgz

Adicionalmente, se necesitará algún programa Windows. La configuración ejemplo, asume que se encontrarán en el directorio */dosc/windows*.

A continuación se analizará el fichero de configuración, el cual es bastante más sencillo que su equivalencia en el emulador DOS. Obsérvese la tabla 3.

En la primera parte, "*drives*", se define la correspondencia entre unidades del emulador y directorios del sistema Linux. A partir de esta sección, es posible escribir el resto de referencias de forma equivalente con un path DOS o Linux.

El segundo apartado, "*wine*", sirve para indicar donde está el directorio de Windows, el de sus librerías, paths de búsqueda y ficheros de recursos. Sólo es posible lanzar programas que se encuentren en el path referenciado aquí.

En "*fonts*" se asignan equivalencias entre los nombres lógicos de fuentes Windows y las fuentes reales X-Windows.

Los dos siguientes apartados, "*serialports*" y "*parallelports*", permiten acceso desde la emulación a módems e impresoras. Nótese que el soporte de apuntador es intrínseco a X, y no se debe permitir acceso al puerto serie donde se encuentre el ratón, si se dispone de tal dispositivo.

El último apartado, "*spy*", controla la información de depuración que puede verse de la sesión del emulador. Es de poca utilidad si sólo se desea usar el emulador, y va orientado más a los interesados en desarrollar partes del sistema.

Lanzar un programa (con X-Windows arrancado) es tan sencillo como ejecutar los siguientes comandos:

cd /dosc/windows

wine winmine.exe

wine pbrush.exe

Con ello se lograrán lanzar un par de aplicaciones para Windows en el terminal X, como se observa en la figura 2. Aunque casi todos los programas que acompañan al mismo Windows ya ejecutan correctamente, las grandes aplicaciones corren con mayor o menor suerte y, desde luego, con menor robustez que sus compañeras en el emulador de DOS. A lo largo de este año, el soporte de programas realmente útiles (procesadores de textos y editores gráficos), de los que Linux aún carece, pueden hacer de éste sistema operativo el "único sistema operativo".

CONCLUSIONES

DOSEMU tiene una meta más definida que *WINE*. Si un programa no funciona en el primer caso, normalmente se debe a una carencia en la implementación del emulador, y las causas de fallo son muy definidas. Mientras que *DOSEMU* imita una arquitectura muy estable en el transcurso del tiempo, el emulador de Windows debe soportar un API menos conocido y sujeto al capricho de continuos cambios en las expectativas de un mercado comercial monopolizado.

A pesar de que las aplicaciones shareware, tienen ya un comportamiento razonablemente bueno con el emulador, las aplicaciones más importantes, las producidas por MicroSoft, que dispone de un total conocimiento de la funcionalidad interna del sistema, encuentran mayor dificultad en ser ejecutadas correctamente. Con la llegada de Windows NT, y posteriormente de Windows'95, las inconsistencias internas que existen entre aplicaciones de 16 y 32 bits dificultan el progreso de *WINE*.

En el futuro, los autores del emulador de DOS pueden llegar a reescribir un sistema operativo, internalizando completamente la funcionalidad que se necesita. Esto tendría el inconveniente de la posible pérdida de compatibilidad, pero con la ventaja de la optimización en ejecución, robustez e integración con el sistema multitarea de Linux.

Mientras que así se aseguraría un abanico de juegos inmenso para Linux, con el perfeccionamiento del emulador de Windows se incorporarían aplicaciones vistosas y profesionales a un robusto, aunque arisco sistema. En algún momento, ambos emuladores pueden llegar a la unificación. Por el momento, las trayectorias de ambos para lograr el objetivo son independientes pero paralelas: ahorrar dinero y esfuerzo a los usuarios. Hasta el mes que viene.

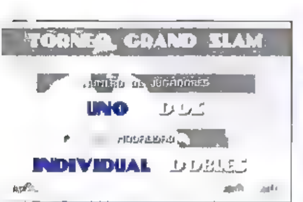
VIRTUAL TENIS

TODO EL TENIS MUNDIAL EN UN CD-ROM DE IMPACTO...

Terrenos de juego creados en 3D con nubes fractales.



Cuadro de partidos disputados en el torneo en curso.



Los torneos se pueden jugar en compañía de un amigo.



Control de los jugadores con teclado o Joystick.



Posibilidad de jugar el Grand Slam completo: permite grabar torneos y temporadas a la mitad.

Virtual TENIS

POR SÓLO 2.995 PTAS. IVA INCLUIDO

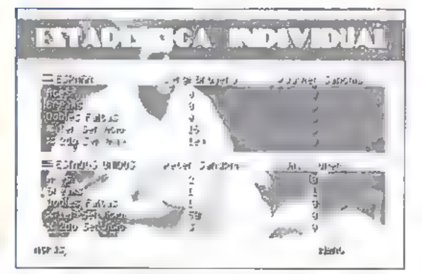
REQUERIMIENTOS : PC 386 o superior, 4MB memoria RAM, lector de CD ROM, tarjeta VGA, disco duro, ratón recomendado. Compatible con Sound Blaster y Gravis Ultrasound.

CONTIENE: PC CD

CON LA GARANTIA DE: DDM DIGITAL DREAMS MULTIMEDIA



Se pueden disputar los cuatro torneos del Grand Slam.



Completas estadísticas de cada partido.



Control independiente de música y sonido.

- Posibilidad de conectarse con distintos jugadores mediante red, módem y cable serie.
- Sistema de menúes totalmente intuitivo.
- Ayuda interactiva durante el desarrollo del juego.
- Fácil instalación en el disco duro.
- Jukebox para escuchar las distintas melodías que contiene.
- Cuatro diferentes superficies.
- Cada uno de los jugadores posee sus propias características.
- Permite elegir el número de sets que se desee disputar.
- Posibilidad de cambiar incluso el detalle gráfico.

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

Solicita VIRTUAL TENIS enviando este cupón o llamando al teléfono (91) 661.42.11 de 9 a 14 y de 15 a 18, o por Fax: (91) 661.43.86

Deseo que me envíen: ☐ **VIRTUAL TENIS por 2995 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos..Domicilio..............................Población..................

Provincia..........................C.P ....F. de nacimiento....................Profesión..............................

FORMA DE PAGO:

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono .... Firma ,

☐ Giro Postal nº..................................de fecha.........................

☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Fecha de caducidad de la tarjeta...............................Nombre del titular, si es distinto..............................

Rellena este cupón y envíalo a: ABETO EDITORIAL C/ Aragoneses, 7. 28100 Pol. Ind. ALCOBENDAS (Madrid)



Las herramientas visuales de Microsoft

Fernando de la Villa

La compañía de Bill Gates ofrece a los desarrolladores de software una completa selección de herramientas visuales denominadas genéricamente Microsoft Visual Tools. En el presente artículo se da un repaso a los productos disponibles actualmente en el mercado pertenecientes a esta familia de herramientas.

El objetivo de las herramientas de programación visuales consiste en facilitar al máximo la ardua tarea del desarrollo de software. Principalmente se busca minimizar el tiempo de desarrollo mediante la aplicación de conceptos de ingeniería de software (como la reutilización de código) y técnicas de diseño (como la orientación a objetos). Sin la ayuda de este tipo de herramientas no sería posible realizar programas complejos y fiables a corto plazo.

Las Visual Tools de Microsoft son un conjunto de herramientas visuales diseñadas para cubrir las necesidades del desarrollador moderno de software. Están formadas por Visual Basic, Visual C++, Visual FoxPro, Fortran PowerStation, Visual SourceSafe y Visual Test. A continuación se exponen, una por una, las herramientas visuales citadas.

MICROSOFT VISUAL BASIC

Desde sus comienzos, Visual Basic ha sido la herramienta preferida por muchos programadores para crear aplicaciones Windows debido a su potencia y gran facilidad de uso. Ya está disponible la versión 4.0, resultado de las experiencias recogidas en base a la versión anterior. De este mismo producto existen tres ediciones claramente diferenciadas: la Edición Estándar, la Edición Profesional y la Edición Empresarial. Cada una de ellas está orientada a un tipo de desarrollador muy concreto.

La Edición Estándar es la elección más adecuada para el principiante. Se trata de una herramienta plenamente integrada en Windows'95 que permite la creación rápida de aplicaciones 32 bits para este sistema operativo y que realiza de forma automática la migración de aplicaciones Visual Basic 3.0 a Visual Basic 4.0. Una de sus características principales es el soporte de acceso a datos. Incluye un sistema de bases de datos integrado y controles para la gestión de datos preparados para ser utilizados directamente en las aplicaciones. Dispone, entre otros, de soporte de datos para Microsoft Access, FoxPro, dBASE, Paradox y Btrieve. Otra de sus características principales es la inclusión de OLE (Object Linking Embedding). Esta tecnología de ensamblaje de componentes ya creados permite la incrustación de objetos funcionales sin necesidad de crear nuevo código ni de realizar pruebas. Además, incluye más de 25 controles estándar, realiza comprobación de sintaxis a medida que se escribe y dispone de un completo depurador de aplicaciones.

La edición Profesional está dirigida a desarrolladores con más experiencia. Incluye todas las características de la Edición Estándar además de otras muchas que amplían su potencia de forma considerable. Entre las nuevas características destacan la posibilidad de crear aplicaciones tanto de 16 como de 32 bits, la compilación condicional para el soporte de múltiples plataformas sin modificar el código fuente y la inclusión de soporte completo de ODBC con controladores de Microsoft SQL Server y Oracle. La Edición Profesional dispone de un nuevo juego de controles que incluye 11 controles OLE basados en Windows'95, más los controles tridimensionales y los controles multimedia. Además, tiene soporte de VBX, controles OLE, servidores OLE y librerías de

enlace dinámico (DLL). Otras características importantes son el compilador de ayudas, el administrador de complementos para la creación de asistentes personalizados y el editor de informes Crystal Reports.

Por último, está la Edición Empresarial, que supone una nueva ampliación de la edición expuesta anteriormente. Su público son los equipos de desarrollo encargados de crear aplicaciones de gran envergadura. Se trata de una herramienta que combina un entorno de desarrollo rápido de aplicaciones con posibilidades distribuidas basadas en la filosofía cliente/servidor. Como es de suponer, las características de esta herramienta se centran en las capacidades de red y de soporte de grupos de trabajo. Entre otras cosas, permite el desarrollo y depuración de aplicaciones distribuidas en un único equipo y dispone de tecnología de automatización remota para el despliegue de aplicaciones distribuidas. Además, incorpora un administrador de componentes que permite el control y despliegue dinámico de componentes OLE. También ofrece seguridad abierta, basada en RPC para controlar el acceso a componentes distribuidos, e incluye La herramienta de control de la versión Visual SourceSafe, que se tratará más adelante.

MICROSOFT VISUAL C++

El lenguaje C++ es uno de los más utilizados a la hora de realizar aplicaciones de gran tamaño debido a su versatilidad. Las características de C++ permiten una cómoda reutilización del código, y la creación de aplicaciones portables y fáciles de mantener. Microsoft Visual C++ proporciona una plataforma profesional de desarrollo de aplicaciones que aprovecha ampliamente las capacidades de este lenguaje.

En el diseño de la versión 4.0 de Visual C++ se ha tenido muy en cuenta el concepto de la reutilización de código. Incluye las MFC (Microsoft Foundation Classes) versión 4.0, formadas por más de 120.000 líneas de código que ponen a disposición del programador más de 150 clases y que permiten aprovechar la interfaz de Windows'95. Con Visual C++ 4.0 se pueden crear aplicaciones para Windows'95, Windows NT, Macintosh y RISC gracias a sus capacidades de desarrollo multiplataforma.

Las Visual Tools son un conjunto de herramientas visuales diseñadas para cubrir las necesidades del desarrollador de software

Visual Basic 4.0 dispone de tres ediciones: Estándar, Profesional y Empresarial

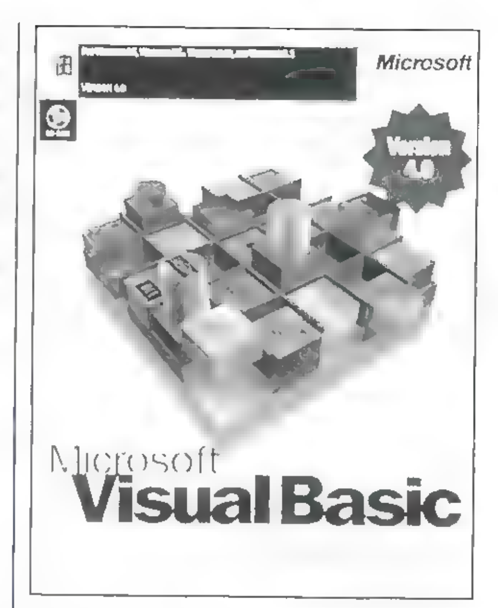
Una de las novedades de Visual C++ es la Galería de Componentes (Component Gallery), un sistema inteligente de almacenamiento y gestión de componentes reutilizables como controles OLE y clases C++. Este sistema permite una fácil reutilización de componentes ya existentes, tanto propios como de otros desarrolladores.

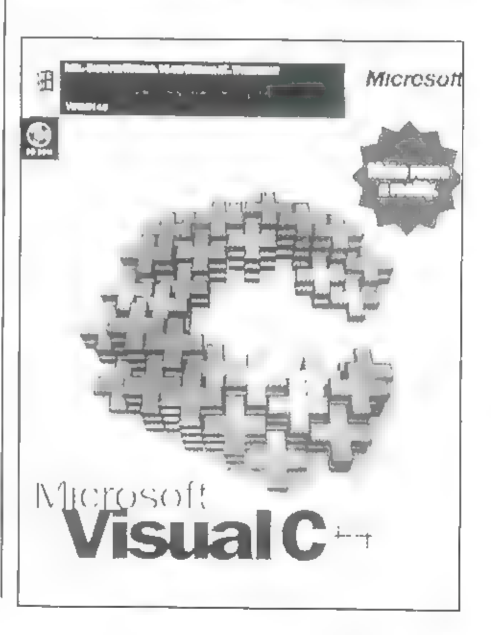
Otra de las características más destacadas de la herramienta son los asistentes personalizados (Custom AppWizards). Con ellos se pueden crear y utilizar plantillas de aplicación para simplificar el desarrollo de programas. Se pueden crear asistentes propios, modificar los ya existentes, e incluso utilizar los de otros fabricantes.

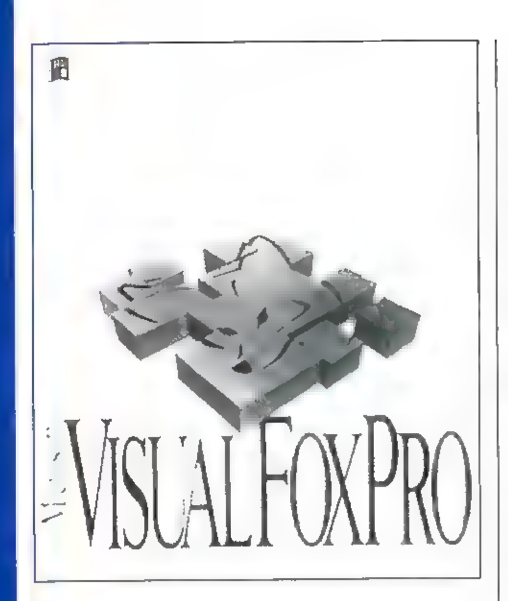
Una posibilidad interesante es la capacidad de ampliación de las Microsoft Foundation Classes mediante librerías de enlace dinámico que derivan clases personalizadas de las ya existentes. Por ejemplo, se podría dar soporte multimedia y soporte a entornos cliente/servidor.

Visual C++ 4.0 soporta las características más avanzadas del lenguaje C++ como los espacios para nombres (namespaces), plantillas, excepciones, reconocimiento de tipos en tiempo de ejecución y la Biblioteca de Plantillas Estándar (STL). Estas características son las que imprimen una mayor potencia a las aplicaciones escritas con C++.

En cuanto al desarrollo de aplicaciones para Windows'95, Visual C++ proporciona un soporte muy completo. Las características más destacadas incluyen encapsulación de nuevos controles







y diálogos comunes de Windows en MFC 4.0, soporte para los nuevos paradigmas del interfaz de usuario como Shell New y soporte para la depuración TCP/IP bajo Windows'95.

Microsoft Developer Studio es otra de las características más notables de la herramienta. Se trata de un entorno centralizado de desarrollo que ofrece integración con múltiples herramientas como la biblioteca para desarrolladores de la MSDN (Microsoft Developer Network), Microsoft Visual SourceSafe, Microsoft Fortran PowerStation y Microsoft Visual Test.

Con ClassView se puede estudiar la estructura de una aplicación en base a sus clases. Permite mostrar las clases, funciones miembro y datos miembro. Además, ofrece la posibilidad de ordenar los miembros por privilegio de acceso y alfabéticamente.

Visual C++ incorpora un completo soporte de acceso a datos. Las clases de objetos de acceso a datos (DAO) proporcionan un acceso rápido orientado a objetos al sistema de bases de datos Jet. Además, se soporta ODBC con Microsoft SQL Server, Microsoft Access y Oracle para la creación de aplicaciones cliente/servidor independientes de la base de datos.

Con ClassView se puede estudiar la estructura de una aplicación en base a sus clases

También existe la posibilidad de apuntarse al programa de suscripción a Microsoft Visual C++. Este programa supone un año completo de suscripción que da derecho a recibir la versión más reciente del producto más tres revisiones periódicas con actualizaciones, información técnica, ejemplos de código y herramientas.

MICROSOFT VISUAL FOXPRO

Microsoft Visual FoxPro es una herramienta visual de desarrollo de aplicaciones de bases de datos relacionales mediante el lenguaje Xbase. La versión disponible actualmente es la 3.0 y supone un gran avance respecto a FoxPro 2.6. La mejora más importante ha sido la inclusión de la orientación a objetos y la adaptación del lenguaje Xbase para soporte de esta característica. La filosofía del producto parte de la idea del desarrollo rápido de aplicaciones (RAD) y una mayor flexibilidad de trabajo que, junto a las técnicas de programación visual, hacen de Visual FoxPro una herramienta muy potente para el desarrollo de aplicaciones de bases de datos relacionales.

El lenguaje C++ es uno de los más utilizados a la hora de realizar aplicaciones de gran tamaño debido a su versatilidad

Visual FoxPro está diseñado para trabajar en entorno Windows y por ello se ha incluido soporte de programación por eventos. De esta forma se consigue que el entorno sea el encargado de gestionar muchos de los eventos producidos. El soporte de programación orientada a objetos incorporado incluye capacidades de herencia, polimorfismo y encapsulación, pero se sigue manteniendo la programación procedural de Xbase.

Visual FoxPro permite al desarrollador una gran interoperabilidad con el uso de objetos y controles OLE, y ofrece una estrecha integración con Microsoft Office y Microsoft BackOffice. Además, soporta los interfaces TAPI y MAPI.

El desarrollo cliente/servidor también se ha tenido cuenta en Visual FoxPro. A través de los controladores ODBC 2.0 de 32 bits se puede acceder, entre otras, a bases de datos Access, Excel, Microsoft SQL Server, Oracle, dBASE y Btrieve. Además, se contempla la posibilidad de ejecución de procesos por lotes (procesos batch).

Uno de los aspectos más cuidados es la conversión de código existente para versiones anteriores de FoxPro. Se puede ejecutar código de FoxPro 2.x para Windows y MS-DOS, y código dBASE para MS-DOS con total facilidad. Además, las pantallas y proyectos de FoxPro 2.x se pueden adaptar a la nueva versión con la ayuda del convertidor incorporado.

La nueva versión del producto incluye una larga lista de asistentes, llamados genéricamente Wizards, que ayudan al desarrollador en las tareas más tediosas. Por poner algunos ejemplos, hay asistentes para importación de datos, análisis y formato del código fuente, acceso selectivo a datos y creación de formularios, gráficos, vistas, consultas y tablas.

Pero el plato fuerte lo constituyen los diseñadores incluidos en el programa. Estas cuatro herramientas visuales reducen sensiblemente el trabajo de desarrollo de una aplicación en Visual FoxPro. El primero de ellos es el gestor de proyectos y sirve para llevar un perfecto control sobre toda la información relacionada con un proyecto. La información se estructura gráficamente en forma jerárquica y se permite el acceso directo a cualquier elemento del proyecto. El segundo diseñador es el de base de datos, utilizado para diseñar cómodamente las tablas, las vistas y las relaciones entre tablas. El diseñador de formularios es el encargado de proporcionar



una manera sencilla y rápida de crear los formularios de la aplicación con los que se enfrentará el usuario de la misma. Admite la utilización de clases de formularios que posteriormente pueden ser reutilizados. El último diseñador es el diseñador visual de clases. Con él se pueden crear propiedades y métodos personalizados.

Visual FoxPro incluye como novedad un diccionario de datos que ayuda a mejorar la integridad y que permite establecer reglas a nivel de tabla que se aplican de forma automática cuando se utiliza esa misma tabla en cualquier lugar de la aplicación.

Existen dos ediciones diferentes de Microsoft Visual FoxPro, la Edición Estándar y la Edición Profesional. La Edición Estándar es más reducida y no incluye algunas de las características del producto como los controles OLE, el asistente para "upsizing" hacia Microsoft SQL Server y el examinador visual de clases.

Visual SourceSafe es un sistema que realiza la gestión de versiones de proyectos de forma automática en Visual Basic y Visual C++

MICROSOFT FORTRAN POWERSTATION

El lenguaje Fortran (FORmula TRANslator) fue diseñado para la creación de programas de tipo científico que realizan cálculos intensivos. Por esta razón se ha venido utilizando principalmente en ordenadores mainframe, poseedores de una gran potencia de cálculo, y en ocasiones muy concretas también se ha utilizado en ordenadores personales. Aprovechando la potencia de los PCs actuales Microsoft ha lanzado Fortran PowerStation, un completo y potente entorno de desarrollo con el lenguaje Fortran.

Microsoft Fortran PowerStation permite el desarrollo de aplicaciones de 32 bits para Windows'95 y Windows NT. Tiene pleno soporte de las características del lenguaje fijadas por el estándar Fortran 90 ANSI/ISO e incluye un compilador ANSI-F77 validado por el Instituto Nacional de Normalización y Tecnología (NIST).

Entre otras características importantes, PowerStation soporta capacidades multi-hilo, multitarea prioritaria y multiproceso en entorno Windows NT.

El lenguaje Fortran fue diseñado para la creación de programas de tipo científico que realizan cálculos intensivos

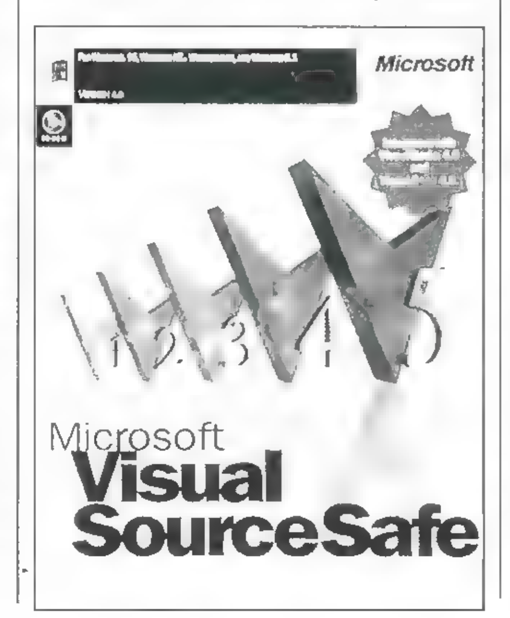
Además, incluye optimizaciones específicas para procesadores Intel 486 y Pentium que permiten una mayor velocidad de trabajo.

La migración del código ya existente se realiza con muy poco esfuerzo. Existen extensiones para IBM VS y SAA y para DEC VAX que permite un fácil traspaso de programas a PC. El paso de código MS-DOS de 16 bits a Windows 32 bits garantiza una total compatibilidad hacia atrás.

Microsoft Fortran PowerStation incluye diversas características en el propio entorno de desarrollo pensadas para mejorar la productividad del desarrollador. Dispone del entorno de desarrollo Developer Studio que integra las diversas herramientas de programación y proporciona interoperabilidad con Microsoft C++ y Microsoft Visual Basic. También incorpora utilidades especializadas en la depuración de programas como DataView y DataTips.

Para aprovechar al máximo las capacidades del lenguaje se proporciona la biblioteca IMSL Fortran, basada en bibliotecas IMSL desarrolladas por profesionales y científicos para grandes sistemas. Esta biblioteca contiene cerca de 1.000 funciones de uso generalizado de tipo estadístico y matemático.

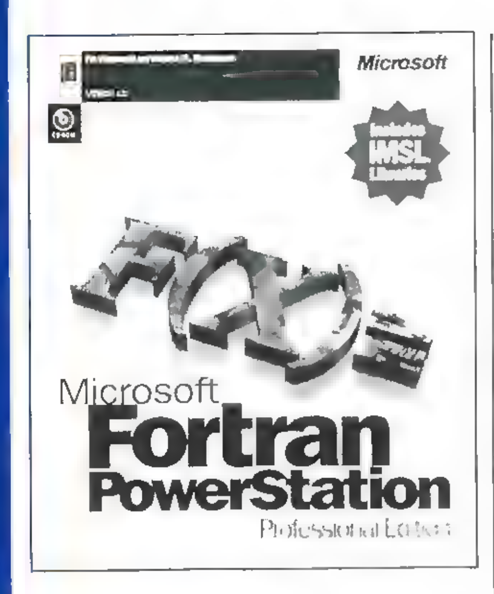
Al igual que otros productos ya comentados, existen ediciones diferentes de PowerStation. La Edición Estándar incluye todas las características básicas y la Edición Profesional añade características más avanzadas como la biblioteca IMSL.



MICROSOFT VISUAL SOURCESAFE

Se trata de un sistema de control de versiones de proyectos que ofrece una completa integración con Microsoft Visual Basic y Microsoft Visual C++. Las funciones principales de Visual SourceSafe incluyen el almacenamiento y organización del código fuente, el seguimiento de las diferentes versiones de un programa y el acceso a los archivos de cada proyecto. Además, incorpora funciones de seguridad y control en un entorno orientado al proyecto.

Los archivos de un proyecto se organizan de tal forma que permiten la realización de modificaciones simultáneas para todos los desarrolladores de un mismo equipo de trabajo. Además crea ficheros históricos con los cambios y ayuda a gestionar el código reutilizable entre proyectos.



SourceSafe permite compartir archivos de forma transparente entre los distintos sistemas operativos utilizados por los desarrolladores. Soporta Windows'95, Windows 3.1, Windows NT Workstation, MS-DOS, UNIX y Macintosh. Para facilitar el trabajo con archivos incluye un explorador similar al incorporado en Windows'95 que gestiona y localiza archivos de forma sencilla.

Actualmente están disponibles versiones para Windows'95, Windows NT Workstation, Windows 3.1, Macintosh (sólo cliente) y MS-DOS. También se puede adquirir una versión para máquinas UNIX a través de MainSoft Corporation con un coste adicional.

MSDN es la fuente oficial de Microsoft para la obtención de información técnica por parte de los desarrolladores

MICROSOFT VISUAL TEST

A la hora de crear un proyecto de software cobra especial importancia la realización de pruebas. Comprobando que el código escrito funciona correctamente se asegura una mayor fiabilidad del producto y se evitan desagradables sorpresas una vez entregada la aplicación al cliente. Microsoft Visual Test es una herramienta especializada en la realización automática de prueba de aplicaciones para Windows'95, NT y 3.1, además de elementos como librerías DLL, componentes DDE y objetos OLE.

Estas pruebas cobran una importancia fundamental en el trabajo en equipo. Un código con errores creado por un programador afectará negativamente a otros programadores que lo utilicen. Visual Test permite detectar los errores individualmente antes de que sea demasiado tarde. Con el fin de minimizar el tiempo de desarrollo y pruebas, soporta tests de manera desatendida fuera del horario de trabajo o incluso mientras se continua escribiendo código. Las ventajas de las pruebas automáticas frente a las manuales son claras: ahorran tiempo y dinero, evitan los errores humanos en la realización de comprobaciones y eliminan los cuellos de botella que se producen en las empresas de desarrollo justo antes de la fecha entrega de la aplicación.

Entre las características de Visual Test destacan la total integración con Visual C++, MSDN y Visual Sourcesafe, la conectividad ODBC, la disponibilidad de Microsoft Developer Studio y la inclusión de la tecnología Boundschecker de NuMega para la detección de pérdidas de memoria y recursos, de pilas y datos dañados, y errores de punteros C++. Además, se incluye la herramienta Microsoft Test 3.0a para probar la corrección de las aplicaciones de 16 bits.

OTROS PRODUCTOS

Aparte de las herramientas expuestas Microsoft proporciona otras utilidades y fuentes de ayuda al desarrollador como Microsoft Developer Network (MSDN) y las Mastering Series.

MSDN es la fuente oficial de información de Microsoft para los desarrolladores con tres niveles de acceso diferentes. El nivel 1 ofrece unos servicios básicos de soporte al desarrollador, que incluyen el envío trimestral de un CD con bibliotecas, ejemplares bimestrales de la revista Microsoft Developer News e invitaciones a acontecimientos especiales de MSDN. El segundo nivel proporciona más beneficios como cuatro actualizaciones trimestrales de la Plataforma de Desarrollo, una colección de compactos con todos los kits de desarrollo, kits de controladores de dispositivo y los sistemas operativos Windows y Windows NT Workstation. Por último, el nuevo nivel 3 está pensado especialmente para el desarrollo y prueba de soluciones basadas en BackOffice.

Las Microsoft Mastering Series son unos cursos basados en el formato CD sobre Visual C++, Visual Basic, Visual FoxPro

Las Microsoft Mastering Series son unos cursos basados en el formato CD sobre Visual C++, Visual Basic, Visual FoxPro y la programación en Access. Estos cursos aprovechan las posibilidades que ofrece la multimedia para facilitar el aprendizaje mediante videos, sonidos y animaciones, y buscan que el desarrollador aprenda practicando con la propia herramienta. Todos ellos incluyen listas con las preguntas más frecuentes (FAQs) , un sistema de índices y un glosario.

PARA MÁS INFORMACIÓN

Se puede obtener más información referente a estos y otros productos contactando con la propia Microsoft, accediendo a la red Microsoft Network o a través del siguiente URL en Internet:

http ://www.microsoft.com



INTRODUCCIÓN A LA GENERACIÓN DE CÓDIGO

Daniel Navarro

Lo que más desagrada a un programador es que le informen de todos los errores y problemas que tiene su programa. El compilador de Letra ya es capaz de hacer ésto. Sin embargo, cuando el programa compilado es correcto, el compilador no es capaz de hacer nada, y eso no está nada bien.

Ahora se tiene que generar código, o dicho de otra forma, se tiene que conseguir que el compilador sea capaz de generar una traducción del programa, pues básicamente es un traductor lo que se está construyendo, es decir, un programa traductor del lenguaje Letra al lenguaje *EML*. En el cuadro 1 se muestran las instrucciones del lenguaje *EML* (a excepción de tres, relacionadas con las funciones especiales, que por ahora no se necesitarán).

El lenguaje *EML* es muy sencillo de generar, pues se diseñó con ese fin. Está formado por una serie de instrucciones independientes (al estilo de un lenguaje ensamblador). *EML* no es ningún lenguaje estándar, sino que es un lenguaje que aquí se ha diseñado específicamente para generar código en Letra; su diseño se basa en el lenguaje *EM* que emplean otros compiladores con el mismo fin. Cuando se construye un compilador para un nuevo lenguaje de alto nivel, se suele diseñar un lenguaje íintermedioí como lo es *EML* para Letra.

En realidad no se va a hacer un traductor, sino dos, ya que el código en *EML* no es ejecutable directamente sobre los PC; después se hará, por tanto, un traductor de *EML* a código máquina 80x86, pero como más adelante se podrá observar, este último proceso de traducción será trivial, ya que en realidad se tendrá en una tabla, para cada instrucción *EML*, una cadena de bytes que será el código máquina 80x86 equivalente a dicha instrucción. Por ejemplo cuando se pretende generar una instrucción *EML* como *IGU* (comparar si dos valores son iguales), en dicha tabla se encuentra la cadena de código 80x86 que implementa un *IGU* en los PC.

Los ílenguajes intermediosí como *EML*, son denominados de esta forma porque se dice que están ía medio caminoí entre el lenguaje fuente (Letra) y el lenguaje destino (código máquina 80x86). Pero en este caso, el lenguaje intermedio resuta que no es ítaní intermedio, pues en realidad está mucho más cerca del lenguaje destino que del lenguaje fuente.

LA MÁQUINA DESTINO

El primer paso que se dará en la generación de código, será definir bien la arquitectura de la máquina para la que se pretende generarlo.

Los PC, según son hoy en día, son máquinas muy complejas. Dentro del sistema operativo DOS se permiten varios modelos de programa, todo el programa y datos en un segmento, en dos, en varios, en modo protegido, etc. El compilador de Letra optará por la primera opción, que es la más sencilla. Si se quiere tener todo el código y datos en un mismo segmento (un máximo de 64k) aún se pueden optar por dos subopciones: un programa *COM* o un programa *EXE*.

Letra generará programas *COM*, aunque Microsoft haya amenazado con dejar de soportarlos (amenaza que son muchos los que dudan que vaya a cumplir). Los programas *COM* son mucho

Generar código es una tarea relativamente sencilla, siempre y cuando no se pretenda generar un 'buen' código (es decir, ni compacto, ni rápido, ni robusto)

más sencillos que los *EXE*, ya que no contienen cabezera (ver cuadro 2). El motivo de dicha decisión es simplemente educativo, el compilador será más fácil de comprender

Un programa *COM* es un fichero que contiene directamente el código y los datos que conforman el programa. Cuando se carga en memoria, lo hace en un único segmento (no importa cual sea) a partir de la dirección 256 (relativa a dicho segmento) y se podría extender hasta la dirección 65535, por lo que su longitud máxima sería de 65280 bytes. No obstante, nunca será tan largo, pues siempre quedará al final de dicho segmento espacio suficiente para la pila de ejecución (estructura que comienza en la dirección 65535 y se extiende hacia direcciones más bajas, ver cuadro 3). En la figura 1 se puede observar dicho modelo de memoria.

El fichero con extensión *.COM* que el compilador genere, no será simplemente una transcripción directa del programa fuente, pues antes de dicha transcripción será incluido en el fichero un código fijo (un comienzo igual para todos los programas que el compilador de Letra genere). Dicho código no es ninguna cabecera, es simplemente la parte inicial del programa que se encargará de las tareas que debe hacer un programa, y que serán ajenas a quienes hagan un programa en Letra. El código fijo inicial contendrá lo siguiente:

- *El código de inicialización.* Es una parte común a todos los programas en Letra que se encarga de cosas como inicializar el módo de vídeo, fijar algunas interrupciones, inicializar alguna variable del sistema, etc.
- *El código de finalización.* Esta parte, aunque se añade dentro de la parte fija al inicio del programa *COM*, se ejecutará cuando se termine el programa, antes de salir al sistema operativo, y se encargará de tareas como restaurar las interrupciones que previamente habia instalado.

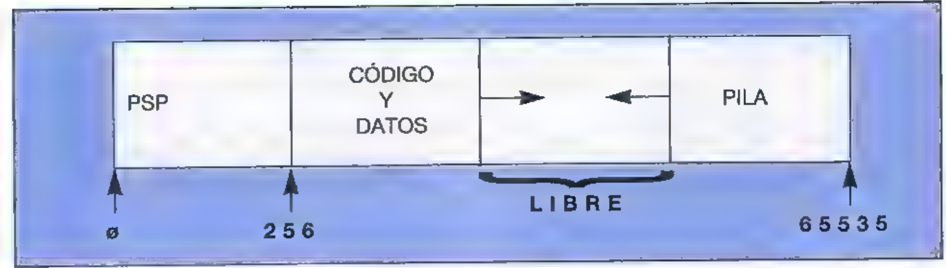


Figura 1. Modelo de un programa COM.

- *El código de las funciones especiales.* Las funciones internas del lenguaje Letra (*mover_cursor, borrar_pantalla, escribir*, etc.) se añadirán aquí también para que el programa generado a continuación pueda llamarlas. Es importante resaltar que cualquier programa compilado en Letra, incluirá el código de todas las funciones especiales del lenguaje, aunque no las utilice.

Todos los compiladores añaden una parte de código como ésta en sus programas compilados, que suelen denominar *startup code*, por eso cuando se compila, por ejemplo, un programa en C como el siguiente:

void main(void) { }

se genera un ejecutable de varios Kb, aunque dicho programa no haga absolutamente nada. Los compiladores actuales emplean librerias de funciones que se añaden al programa en la fase de enlazado (*linking*), Letra no tiene ninguna fase de enlazado, sino que genera directamente el fichero ejecutable.

Los compiladores usan lenguajes intermedios como EM para generar código fácilmente

UN EJEMPLO BÁSICO

En realidad, lo único que interesa de todo esto, es saber que se va a generar el código destino en una tabla de 64kb (65536 bytes), que a partir de la posición 256 se va a precargar el código *startup* (que proviene de un fuente en C que se estudiará más adelante), a continuación del mismo generará la traducción (directamente del código *EML*) del programa compilado. Al finalizar el proceso se grabará dicha tabla, pero desde la posición 256 hasta la última posición que haya sido empleada por el programa compilado, en un fichero con el mismo nombre que el programa fuente pero con extensión ".com", siendo así un fichero ejecutable por DOS.

Si se compila, por ejemplo, el siguiente programa en Letra en un fuente denomindado "prueba.lt".

VOLVER ;Solo esto

El compilador actua de la siguiente forma:

1.- Inicializa el vector de generación de código *mem[65536]*, que representa la memoria de la máquina destino (el segmento donde se ejecutará el fichero *COM*).

2.- Carga en dicho vector el código fijo inicial o *startup* (que está contenido en un fichero que ya se estudiará) a partir de la dirección 256 y obtiene, en una variable denominada *IP*, la siguiente dirección de memoria disponible para generar código. Por ejemplo, si el *startup* ocupa 3000 bytes, entoces *IP* será 3256.

3.- Carga el programa en memoria y obtiene la primera pieza (*p_volver*).

4.- Comienza el análisis al ser invocada la función *an_programa()*. Ésta, a su vez, llama a la función *an_sentencia()* y ésta a *an_sent_volver()*. Esta función será la encargada de generar el código

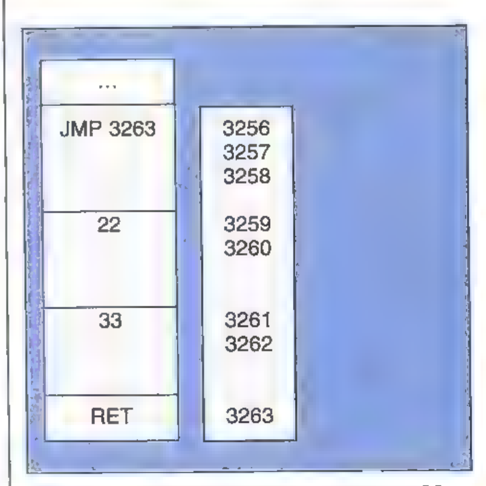
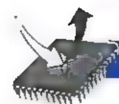


Figura 2. Programa compilado en mem[ ].



para un volver, y para ello en *EML* se tiene la instrucción *RET* que significa volver. En otra tabla se tenadrá el código al que equivale dicha instrucción, que será escrito en el vector *mem[ ]* a partir de la posición *IP* (en este caso, el código para un *RET* ocupa un solo byte). Se actualiza *IP* a la siguiente posición de *mem[ ]* disponible.

5.- Se obtiene la siguiente pieza que es *p_eof* (fin del fichero). Las funciones de análisis retornan y finaliza el proceso de análisis.

6.- La compilación ha acabado con éxito, se graba el vector *mem[ ]* desde la posición 256 hasta la posición 3256 incluidas, con una longitud total de 3001 bytes (*startup* + código de la instrucción *RET*) en un fichero denomindado "prueba.com".

GENERACIÓN DE DATOS

En Letra, los datos (variables) se generan de forma muy similar al código, ya que en este lenguaje, una declaración de datos se comporta como una sentencia, en cuanto a que pueden declararse datos en cualquier parte del programa. Si se tiene el siguiente programa en Letra:

DATOS
a=22
b=33
FIN
VOLVER

Se reservará en el vector de memoria destino (*mem[ ]*) y a partir de la posición actual (*IP*), el espacio requerido para almacenar los datos declarados. En Letra, cada dato es implementado como un número entero de 16 bits con signo (2 bytes), por lo que, para la declaración del ejemplo anterior, se reservarían 4 bytes.

Además, se genera antes de cada zona de declaración de datos un salto al final de dicha zona, para evitar que se intenten ejecutar los datos (cosa que, en el mejor de los casos, bloquearía el ordenador). Dicho salto se genera mediante la instrucción *EML JMP* a la que, como parámetro, se le indica la dirección (dentro de *mem[ ]*) a la que debe saltar. En la figura 2 se puede observar como quedaría el programa una vez compilado, teniendo en cuenta que *JMP* representa a una determinada cádena de código 80x86.

CUADRO 1

Se describen a continuación las instrucciones básicas del lenguaje intermedio *EML* utilizado para generar código en el compilador de Letra. Se detalla el mnemónico de la instrucción y tras él (entre los signos < y >) el parámetro requerido en caso de que lo tenga. A continuación se describe técnicamente su fucionamiento, teniendo en cuenta lo siguiente: los corchetes [ ] indican la referencia (dentro del vector *mem[ ]*) al dato que está en la dirección indicada entre los corchetes. Tras esta descripción técnica, se ofrece una descripción más detallada del funcionamiento de la instrucción.

INSTRUCCIONES DE ACCESO A DATOS
LOC <constante>: apilar constante;
Apila la constante pasada como parámetro.
LOE <dirección>: apilar [dirección];
Apila la variable que está en la dirección indicada.
LOF <dirección>: desapilar a; apilar [dirección+2*a];
El parámetro pasado indica la posición de inicio de una tabla. Primero se desapila el índice y se multiplica por dos. Después se apila el dato que está en la tabla en la posición que indicaba el índice (el dato que está en *inicio_tabla+2*indice*).
STE <dirección>: desapilar [dirección];
Desapila una variable, es decir, saca un valor de la pila y lo guarda en la dirección que se le indica.
STF <dirección>: desapilar a; desapilar b; [dirección+b*2]=a;
El parámetro pasado indica la posición de inicio de una tabla. Primero se desapila un valor, entonces se desapila el índice y se multiplica por dos, y, por último, se guarda el valor en la tabla en la posición indicada por el índice (se desapila un dato en *inicio_tabla+2*indice*).

INSTRUCCIONES ARITMÉTICAS BINARIAS
MUL: desapilar a; desapilar b; apilar b*a;
Desapila dos valores y después apila el producto de ambos.
DIV: desapilar a; desapilar b; apilar b/a;
Desapila dos valores y apila el resultado de dividir el último por el primero.
MOD: desapilar a; desapilar b; apilar b%a;
Desapila dos valores y apila el resto de la división anterior.
SUM: desapilar a; desapilar b; apilar b+a;
Desapila dos valores y apila su suma.
RES: desapilar a; desapilar b; apilar b-a;
Desapila dos valores y apila el segundo menos el primero.

INSTRUCCIONES BINARIAS
NEG: desapilar a; apilar -a;
Desapila un valor, lo cambia de signo y lo vuelve a apilar.
NOT: desapilar a; apilar not a;
Desapila un valor, lo niega a nivel binario (pasa los unos a ceros y viceversa) y entonces lo vuelve a apilar.

INSTRUCCIONES LÓGICAS
AND: desapilar a; desapilar b; apilar b and a;
Desapila dos valores y apila el resultado de la operación íandí binaria de ambos (opera todos sus bits con la siguiente lógica: *0 and 0=0, 0 and 1=1, 1 and 0=1, 1 and 1=0*)
OR: desapilar a; desapilar b; apilar b or a;
Desapila dos valores y apila el resultado de la operación íorí binaria de ambos (lógica: *0 or 0=0, 0 or 1=1, 1 or 0=1, 1 or 1=1*)

INSTRUCCIONES DE COMPARACIÓN
DIS: desapilar a; desapilar b; si (b!=a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si son distintos, mete en la pila una constante que indique *CIERTO*. En caso contrario mete una que indique *FALSO*. En Letra el valor lógico cierto está representado por cualquier constante numérica que sea impar (que su bit nº0 esté a 1) y el valor lógio falso por cualquier constante numérica par (que su bit nº0 esté a 0).
IGU: desapilar a; desapilar b; si (b==a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si son iguales, mete en la pila una constante que indique *CIERTO* (cualquier número impar). En caso contrario mete una que indique *FALSO* (cualquier número par).
MAY: desapilar a; desapilar b; si (b>a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si el último es mayor que el primero, apila una constante que indique *CIERTO*. En caso contrario apila una constante que indique *FALSO*.
MAI: desapilar a; desapilar b; si (b>=a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si el último es mayor o igual que el primero, apila una constante que indique *CIERTO*. En caso contrario apila una constante que indique *FALSO*.
MEN: desapilar a; desapilar b; si (b<a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si el último es menor que el primero, apila una constante que indique *CIERTO*. En caso contrario apila una constante que indique *FALSO*.
MEI: desapilar a; desapilar b; si (b<=a) apilar CIERTO; sino apilar FALSO;
Desapila dos valores y, si el último es menor o igual que el primero, apila una constante que indique *CIERTO*. En caso contrario apila una constante que indique *FALSO*.

INSTRUCCIONES DE CONTROL DE FLUJO
IR <dirección>: ir dirección;
Es un salto incondicional (como un *JMP* o *GOTO*) a la dirección indicada como parámetro. La próxima instrucción que se ejecute será la que comience en *mem[dirección]*.
IRF <dirección>: desapilar a; si (not a) ir dirección;
Desapila un valor y, si éste es *FALSO* (es un número par), salta a la dirección indicada.
LLA <dirección>: llamar dirección;
Llama (hace un *CALL* o *GOSUB*) a la función que está en la dirección indicada, es decir, apila la dirección de retorno (la dirección de la instrucción siguiente a *LLA*) y entonces salta a la dirección.
RET: retorno;
Saca de la pila una dirección (apilada por un *LLA*) y hace un salto incondicional (como *IR*) a dicha dirección.

El lenguaje intermedio EML.

Se debe recordar que, cuando el compilador se encontraba una declaración de un dato, guardaba en la tabla de objetos un registro con información como tipo de objeto, nombre, dirección, etc. En el campo referente a la dirección del objeto ahora se indicará su posición dentro del vector *mem*. Si en el programa anterior, el código *startup* ocupaba 3000 bytes (de la dirección 256 a la 3255), y la instrucción *JMP* ocupaba 3 bytes (de 3256 a 3258), entonces las variables *a* y *b* serán registradas en la tabla de objetos como sigue:

tipo_objeto=dato, nombre="a", dirección=3259, ...

tipo_objeto=dato, nombre="b", dirección=3261, ...

y la instrucción *JMP* saltará por tanto a la dirección 3263, que es la primera posición libre de *mem[ ]*, tras la declaración de datos.

Durante la compilación del programa, las posiciones del vector de memoria destino correspondientes a los datos *a* y *b* (de 3259 a 3262) serán inicializadas de forma que valgan 22 y 33 respectivamente. Cada vez que en el programa se utilice el dato *a* o *b* se estará con ello haciendo referencia al entero de 16 bits que está en la posición de vector que indican los registros de ambos objetos (para encontrar la dirección de memoria a la que se refiere una variable como *a*, el compilador no tiene más que buscarla en la tabla de objetos y mirar su registro). Por ejemplo, poner en el programa la siguiente sentencia:

a=100

Se traducirá en las instrucciones *EML* necesarias para poner un 100 en el dato contenido en *mem[3259]*.

Se debe recordar que en Letra, además de los datos, se manejan tablas de datos; cuando se declara una tabla de datos como la siguiente:

mi_tabla[4]=100:101:102:103:104

Suponiendo que *IP* vale 8192 cuando se está declarando la tabla, se creará un objeto como el siguiente:

tipo_objeto=tabla, nombre="mi_tabla", dirección=8192, longitud=5, ...

Teniendo en cuenta que la longitud de *mi_tabla[4]* es de 5 (de 0 a 4) datos de 2 bytes cada uno, *IP* valdrá 8202 tras declarar la tabla.

El acceso dentro del programa a una tabla es algo más complicado. Si, por ejemplo, se pone dentro de un programa:

mi_tabla[a]=1

se tienen que generar las instrucciones *EML* necesarias para que en la dirección referida por *mi_tabla[a]* se guarde un 1. Dicha dirección se calcula como: *Inicio de la tabla (8192) + 2 * \|a\| (contenido en mem[3259]).*

Luego, como se puede ver, para acceder a una tabla se calcula el valor del índice, se multiplica por dos, y se suma a la dirección inicial de la tabla.

Letra permite acceder a un dato o a una tabla antes de que sean declarados. Esto plantea el problema de cómo generar el código para acceder a ellos, cuando aún no se sabe en qué dirección van a ser ubicados (hasta que sean declarados), dado que el compilador funciona en una sola pasada. La resolución a este problema no es trivial, por lo que será resuelta con detenimiento posteriormente. Hasta entonces, es mejor obviar esa situación, y suponer que todos los datos se declaran antes de ser usados por el programa.

CUADRO 2

Los programas *COM* son, simplemente, un volcado de memoria en disco, no contienen ni cabecera ni ningún tipo de información adicional como pudiera ser:

- Información de la posición del código, de los datos y de la pila. El código y los datos se cargan juntos a partir de la posición 256 del segmento, tal cual están en el fichero *COM* del disco. El puntero de pila siempre es fijado (por DOS, antes de ejecutar el programa) de forma que apunte a la última palabra dentro del segmento.
- Información sobre reubicación. No es necesario reubicarlos, es decir, si el compilador calcula que una variable va a estar en la posición de memoria 383, estará en dicha posición siempre que el programa se cargue, en cualquier ordenador y en cualquier situación. Esto es porque los PC (en modo real, no en modo protegido) manejan las direcciones de programa en dos partes: *segmento* y *offset*. El *segmento* dependerá de en qué parte de memoria se cargue el programa. Por lo tanto, variará según el ordenador y las condiciones en las que se ejecute el programa, pero el compilador de Letra no hará ningún uso de los segmentos. Sin embargo, el *offset* (que para el compilador, será el puntero real del programa) siempre hará referencia a direcciones relativas dentro de dicho *segmento*. Luego, según el *offset*, el programa siempre comenzará en la dirección 256 y la variable anterior. Por tanto, siempre estará en la posición (*offset*) 383.

Cuando el DOS ejecuta un programa *COM*, realiza los siguientes pasos:

- Busca un *segmento* libre y crea en el mismo, entre las posiciones 0 y 255 el *PSP*. Ésta es una área con información del DOS sobre el programa. Por ejemplo, a partir de la dirección 129 se encuentran los parámetros que se le hayan pasado al programa (por ahora el compilador de Letra no ofrece ninguna posibilidad de acceder a dicha información).
- A continuación, carga el programa en dicho *segmento* a partir de la posición 256 (tras el *PSP*).
- Entonces el DOS asigna a los principales registros de *segmento* (*CS*, *DS*, *ES* y *SS*) la dirección del *segmento* donde se encuentra el programa; a partir de este momento todas las direcciones (offset) harán referencia a direcciones relativas dentro del mismo.
- Y por último el DOS fija el puntero de pila en la dirección 65534 y comienza ejecutar el programa por la dirección 256.

Sobre los programas COM.

LAS EXPRESIONES

La parte más complicada de entender dentro del proceso de generación de código son las expresiones. En el análisis sintáctico se realizó la definición de una expresión: básicamente se puede definir con la siguiente regla en notación BNF:

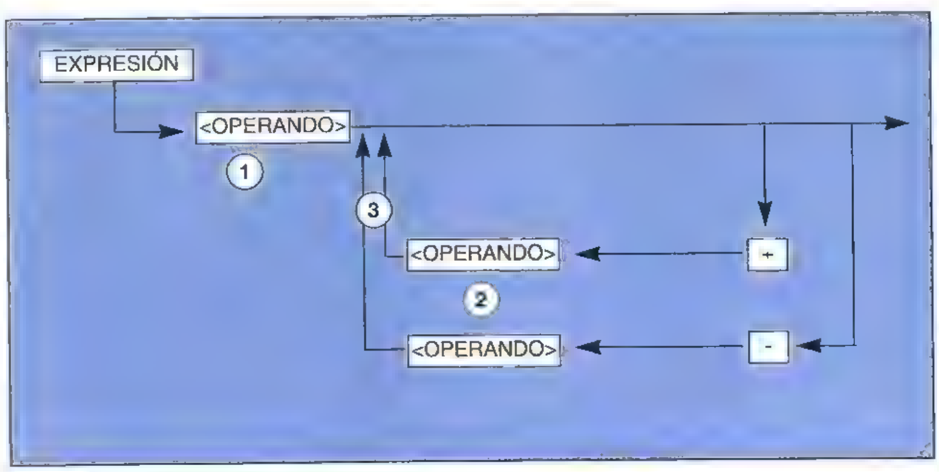
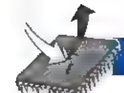


Figura 3. Grafo del analizador de expresiones (+,-).



<EXPRESIÓN> ::= <OPERANDO> { <OPERADOR> <OPERANDO> }

Por *<operando>* se entiende una parte de la expresión como puede ser una constante, un dato, otra subexpresión entre paréntesis, etc., y por *<operador>* cualquier operador binario que realice una operación entre dos operandos (suma, resta, multiplicación, etc.). La expresión anterior quiere decir que una expresión es algo que comienza con un operando y tras él, mientras venga un operador seguido de otro operando, continúa la expresión.

Si, por ahora, se admite como operandos, simplemente constantes numéricas y como operadores sólo la suma y la resta, puede tenerse expresiones como las siguientes:

5
33+2
33-1+2
0-99+1+99

El código del analizador de estas expresiones está representado en el gráfo de la figura 3 y se corresponde con el siguiente procedimiento de análisis (se puede observar la correspondencia directa entre la regla BNF, el grafo y el procedimiento de análisis).

```
void an_expresion(void) {
if (pieza_actual!=p_entero) error(p_entero);
/* 1 */
obtener_pieza();
while (pieza_actual==p_suma || pieza_actual==p_resta) {
obtener_pieza();
if (pieza_actual!=p_entero) error(p_entero);
  /* 2 */
  /* 3 */
  obtener_pieza();
 }
}
```

Si se desea generar código para que en tiempo de ejecución se calcule dicha expresión, se debe hacer dentro de los procedimientos de análisis, teniendo en cuenta que cada vez que se genere código, será sobre el vector *mem[ ]* a partir de la posición indicada por *IP*.

EJECUCION SOBRE PILA

El lenguaje *EML* no utiliza registros, como los lenguajes ensambladores, sino que hace todos los cálculos sobre la pila. Cuando se evalúa una expresión, se tiene que hacer de forma que, al acabar, el resultado de la expresión quede en la cima de la pila.

Por ejemplo, si se desea evaluar la expresion *2+3*, se utilizará la instrucción *LOC* (Ver su función en la descripción del código *EML*, cuadro 1) para apilar la variable 2. Tras ello, y con un *LOC 3*, se apila la variable *3*, tras lo que se tiene en la cima de la pila los dos valores a sumar. Entonces, *EML* proporciona operadores binarios como *SUM* (suma) que operan con los dos valores de la cima de la pila y dejan en ella el resultado, luego la siguiente secuencia de código *EML*:

LOC 2; LOC 3; SUM;

Calcula la expresión *2+3* y, por tanto, dejará, después de ejecutarse, un 5 en la cima de la pila. El proceso se muestra en la figura 4.

Tanto en el grafo de la figura 3 como en el procedimiento *an_expresión()* visto anteriormente, hay marcados 3 puntos: en el primero es donde se debe generar el primer *LOC*, en el segundo el segundo *LOC*, y en el tercero es donde se debe llamar a la función que genere el operador correspondiente (en el ejemplo anterior puede ser *SUM* (suma) o *RES* (resta).

Una expresión que involucre a los dos operadores como puede ser la siguiente:

3-99+8-1

CUADRO 3

La pila es una estructura de datos formada por una lista de palabras (una palabra es un número entero de 16 bits, o dos bytes) que mantiene siempre un puntero a la última palabra de dicha lista, y permite realizar únicamente dos acciones:

- Apilar otra palabra (se añade otro entero a la lista), para ello se resta 2 (dos bytes = una palabra) al puntero de pila y se pone la nueva palabra en la dirección apuntada ahora por el mismo.
- Desapilar una palabra, se obtiene la última palabra que se apiló, y después se suma 2 al puntero de pila, con lo que apuntará a una palabra anterior a la desapilada (apuntará a una que fué metida antes en la pila).

La pila de ejecución de un programa COM comienza al final del segmento del programa (offset 65534); cuando se apilan valores la pila crece hacia direcciones más bajas y se debe por tanto, tener cuidado para que no llegue a colisionar con el código y los datos, lo que produciría un desbordamiento de pila (o ¡Stack overflow!), y podría producir un resultado catastrófico. Otros compiladores pueden dotar a los programas de mecanismos para prevenir un desbordamiento de pila e informar cuando esto suceda, pero el compilador de Letra no lo hace.

La pila de ejecución se utiliza generalmente para, cuando se llama a una función, guardar la dirección a la que esta debe retornar al terminar; cuando la función termina, desapila la dirección de retorno y devuelve el control a su llamante. Se utiliza una pila para dicha labor porque permite que desde una función se llame a otra función, que a su vez llame a otra ..., pues se pueden apilar muchos valores en la pila y luego, a medida que las funciones vayan retornado unas a otras, se irán desapilando.

La pila se utiliza además, por muchos lenguajes, como el medio de pasar parámetros a una función, apilando estos antes de la dirección de retorno; la función accederá directamente a la pila en busca de estos valores

En los programas compilados de Letra se utiliza además la pila como almacenamiento temporal para calcular expresiones (en vez de los registros del procesador); una vez finalizado el cálculo de la expresión solo quedará apilado el resultado de la expresión, que será inmediatamente desapilado por el programa antes de continuar su ejecución.

La pila de ejecución.

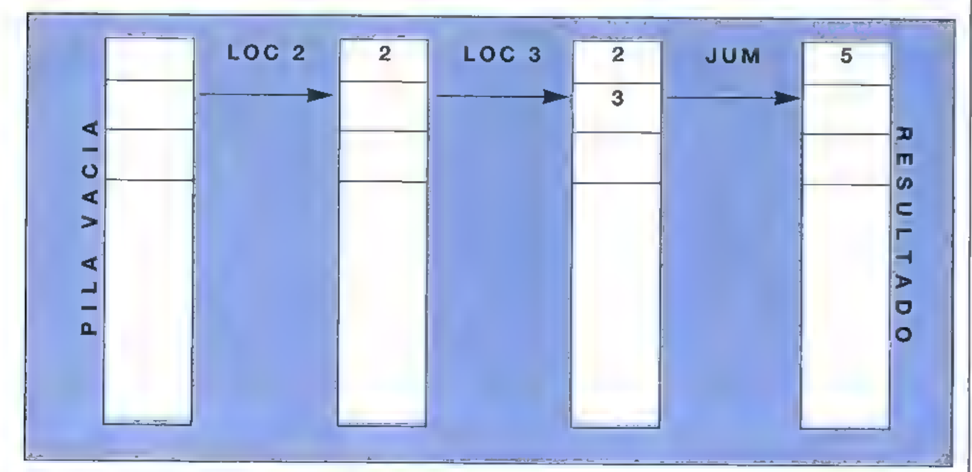


Figura 4: Evaluación de la expresión 2+3.

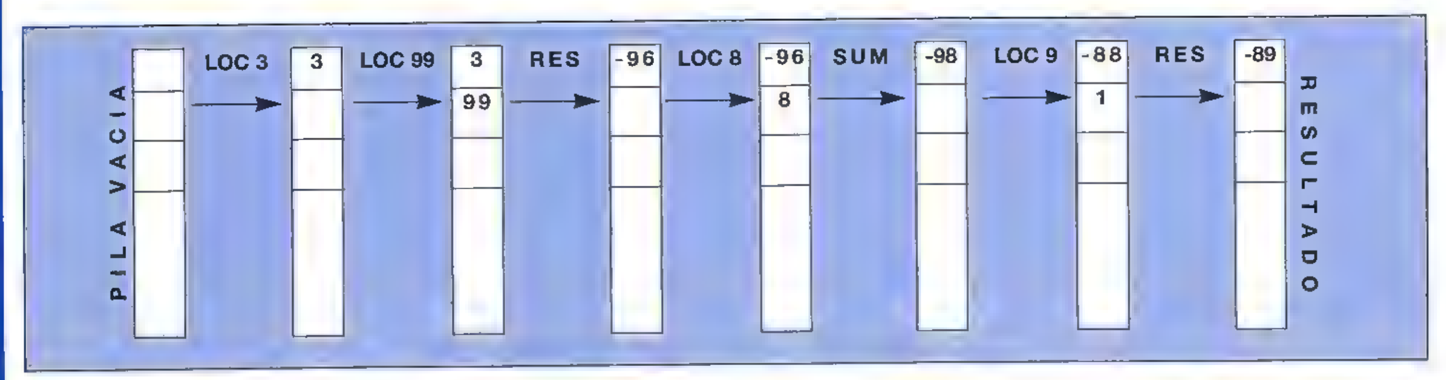


FIGURA 5: Evaluación de la expresión 3-99+8-1.

Será convertida por el compilador en la siguiente secuencia de código (esto se puede observar siguiendo la lógica de la funcion *an_expresión()*).
LOC 3; LOC 99; RES; LOC 8; SUM; LOC 1; RES;

Esta forma de poner la expresión (3 99 - 8 + 1 -) se denomina "notación polaca inversa". Ahora es en la figura 5 donde se muestra el proceso de evaluación.

EXPRESIONES MÁS COMPLEJAS

El modelo de expresión que se ha expuesto puede ser ampliado con más operadores, además de la suma y la resta, pero si se pretende con dicho modelo evaluar la expresión siguiente:
*2+10*3*

Teniendo en cuenta que la instrucción del lenguaje intermedio que efectúa el producto de dos números es *MUL* (multiplicar), se generará el siguiente código *EML* por el compilador:
LOC 2; LOC 10; SUM; LOC 3; MUL;

Lo que dará al evaluarse un resultado erróneo de 36 (resultado de (2+10)*3), en lugar de 32 (2+(10*3)). Lo que ocurre es que no se ha definido un modelo que contemple distintos niveles de prioridad en los operadores, luego las operaciones se realizarían con este modelo de izquierda a derecha, sin importar de que operadores se trate.

Evidentemente, ésto no es lo que hacen los lenguajes de programación, es más, el modelo de *<EXPRESIÓN>* que en los capítulos anteriores se habia definido para Letra SÍ que contemplaba dicha prioridad, pero ya que no se dispone de más espacio, será en el próximo artículo donde sea resuelta la generación de código con operadores de varios niveles, operadores unarios, acceso a datos y tablas, etc.

Por ahora se podría pensar en forzar la prioridad de los operadores utilizando paréntesis (subexpresiones) para indicar qué cálculos se deben realizar antes que otros. Para permitirlo, simplemente se tiene que implementar un modelo de expresión, como el definido anteriormente, pero en lugar de definir un operando únicamente como:
<OPERANDO> ::= p_entero

Se tiene que definir como una de estas dos opciones:
<OPERANDO> ::= p_entero | p_abrir_paréntesis <EXPRESION> p_cerrar_paréntesis

Es decir, que se admite como un operando un simple número, o bien otra expresión completa. A continuación se detalla el modo de funcionar del proceso de generación de código para este modelo.

Cuando se está analizando una expresión, es invocada la función *an_operando()*, y ésta puede actuar de dos formas:
- Si viene un número entero, entonces la función genera código para apilar dicho número con un *LOC* y retorna.
- Si viene una pieza de tipo *p_abrir_paréntesis*, se obtiene la siguiente y se llama al proceso *an_expresion()*. Este retornará cuando se acabe la subexpresión, habiendo antes generado el código necesario para calcularla y dejar su resultado apilado.

Por supuesto, este modelo permite que dentro de una subexpresión haya otras subexpresiones. Cada *p_cerrar_paréntesis* terminará la última subexpresión comenzada. Luego una expresión como la siguiente:
*2+(10*3)*

Será traducida por el compilador como la siguiente cadena de código:
LOC 2; LOC 10; LOC 3; MUL; SUM;

Como se puede observar, cuando llega la subexpresión, se calcula por completo primero (*LOC 10; LOC 3; MUL;*) y después se retorna a la expresión principal.

Como es importante que estas bases queden claras, se dará un último ejemplo un poco más complejo de una expresión, y el código que el compilador generaría con el modelo estudiado. Se invita al lector a que intente obtener el código a partir de la expresión antes de observarlo:
*(14/2*32)-((93-25*40)+(8*9)-15+7)/(19-29)*
LOC 14; LOC 2; DIV; LOC 32; MUL; LOC 93; LOC 25; RES; LOC 40; MUL; LOC 8; LOC 9; MUL; SUM; LOC 15; RES; LOC 7; SUM; RES; LOC 19; LOC 29; RES; DIV;

El resultado de dicha expresión (evaluada de esta forma, y no de la forma que la evaluaría un lenguaje con prioridad de operadores) es 256 ... ¿verdad?.

PRÓXIMO NÚMERO

En este número, el artículo no va acompañado por ninguna implementación en el disco, ya que se trataba únicamente de introducir el proceso de generación de código. Quedan aún muchos detalles de éste por aclarar, que se irán viendo poco a poco según se vaya construyendo el generador.

En el próximo número, se abordará de lleno la generación de código para expresiones complejas y se implementará en el compilador.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS

CÓMO SUSCRIBIRSE A



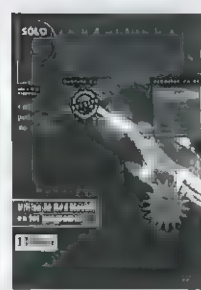
PROGRAMADORES

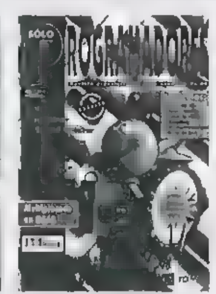
Revista práctica para usuarios de PC

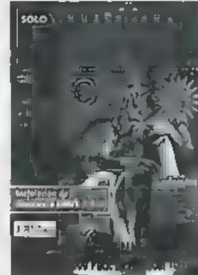


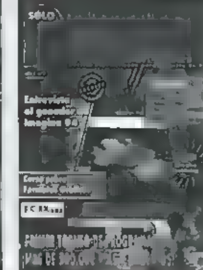




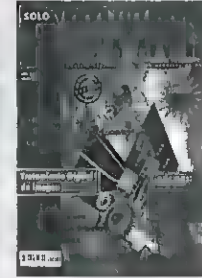






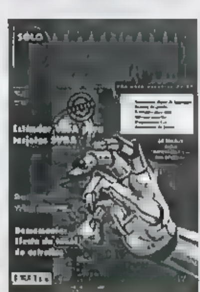




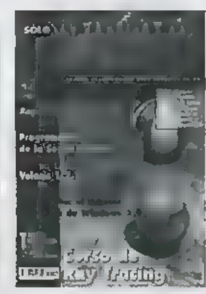


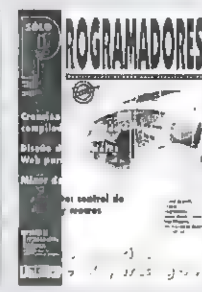














Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos...Domicilio...

Población.................................C.P ....Provincia.....................Telf.........................Profesión..................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto.................................

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número..

el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.
como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)



INTRODUCCIÓN Y CONCEPTOS BÁSICOS

María Jesús Recio

Posiblemente, los lectores se hayan preguntado alguna vez cómo funciona una red, ya que, incluso parece misterioso introducir una hoja de papel en un "aparatito" y que aparezca una copia a miles de kilómetros de distancia

Hace algunos años, cuando se planteaba el diseño de una nueva aplicación, no se pensaba en la necesidad de incorporar facilidades para posibilitar la comunicación por red de dicha aplicación. En la actualidad, cuando alguien se plantea el crear un nuevo programa, debe planificar la incorporación de funciones de red, si quiere que su creación sea competitiva. Esto pone de manifiesto la necesidad de acercarse al mundo de las redes, que ya no permanece aislado en el sector del hardware. De esta forma, todo programador que se precie no puede dejar en manos de los "hardwareros" el que le arreglen "esos pequeños problemas de red" tan inoportunos, sino que debe ser capaz de conocer cómo funciona, cuales son sus secretos y sus posibilidades, y familiarizarse con todos los términos de este mundo que hasta ahora parecían lejanos.

Realice un pequeño examen de conciencia y contéstese a estas cuestiones: ¿Cuantas veces se ha preguntado en qué consiste el maravilloso mundo de las redes,? qué es eso de protocolos de comunicaciones, topologías de redes, servidores, arquitectura cliente/servidor, sistemas operativos de red, compartición del recursos, interconexión de redes...?

El mundo de las comunicaciones es fascinante, pero oscuro. Existen muchos conceptos que son necesarios conocer para poder comprender mejor este mundo, y así adentrarse en él.

Esta serie de artículos pretende que el lector se adentre en el mundo de las redes, conozca gran parte de la terminología adherida a él, así como aclarar conceptos, y malentendidos. En el presente artículo se pretende hacer una breve introducción, explicando de forma sencilla e introductoria, los conceptos más destacados.

Una red es un conjunto de dispositivos conectados entre sí y cuya finalidad es la compartición de recursos. De esta definición se deduce que no solamente existen redes de ordenadores, aunque sean éstas de las que se ocupará este curso.

Para conectar varios ordenadores se necesita disponer de un medio físico que establezca dicha conexión. Este medio físico suele ser un cable (que puede tener diferentes naturalezas), o el aire. Además del cable, es necesaria la presencia de unos dispositivos (tarjetas) que controlen de alguna forma la comunicación, junto con el software pertinente. Sólo con esto, ya se puede tener una red.

Para tratar de forma global el tema de las redes, se debe hablar de los siguientes elementos: topología, arquitectura, medio físico, método de acceso al medio, protocolos, tarjeta de red, sistemas operativos de red, e interconectividad de redes. Poco a poco se ira profundizando en cada uno de ellos.

Las redes se catalogan de acuerdo al espacio físico por el que están distribuidas, aunque también existen otros modelos de clasificaciones. Una red de área local (*LAN*) es una red cuyos componentes se encuentran dentro del mismo área, por ejemplo un edificio. Una red de campus es una red que abarca varios edificios dentro del mismo área (polígono industrial, un campus, etc.). Una red de área metropolitana (*MAN*) es una red que se extiende por varios edificios dentro de

una misma ciudad. Cuando se habla de una red de área extensa (*WAN*) se está haciendo referencia a una red que abarca diferentes ciudades e incluso diferentes países. Por último, una red corporativa es una red que interconecta todos los equipos de una Organización (empresa), independientemente del lugar en el que se encuentren.

Quizá el lector se está preguntado cómo es posible conectar ordenadores que se encuentran a grandes distancias, como es el caso de redes de área extensa. Habitualmente, en estos casos se utilizan medios como los satélites, la telefonía, etc. Esto significa que se puede alquilar una línea telefónica y utilizarla como "cable" de la red, aunque siempre existe la posibilidad de tirar un cable de un ordenador hasta otro, salvando montañas, edificios, lagos, etc.

¿Por que se ha hecho tan necesaria la creación de una red en cualquier empresa?. Hasta no hace mucho tiempo, no se tenia la necesidad de conectar varios ordenadores. Cada usuario trabajaba en su equipo y cuando necesitaba transportar información, lo hacia en un disquete. En la actualidad, el trasiego de información es muy importante, por lo que la idea de llevar y traer un disquete se hace algo "engorrosa". Eso, sin tener en cuenta que el tamaño de la información que se maneja suele ser bastante mayor de lo que cabe en un disquete. Este es el primer motivo que se puede pensar acerca de la necesidad de las redes. Entrando un poco más en materia se pueden citar motivos de mucho más peso, como pueden ser:

- Compartición de recursos software: Es mucho más barato comprar una aplicación para una red de 20 ordenadores, que comprar 20 aplicaciones.
- Compartición de recursos hardware: Evidentemente, también resulta más barato comprar un par de impresoras, un *plottter*, y un *streamer* para la misma red permitiendo que los 20 ordenadores lo utilicen que comprar uno para cada equipo.
- Compartición de bases de datos: Para entender esta ventaja, solamente es necesario pensar en unos grandes almacenes que tienen muchos ordenadores como cajas registradoras. Cada ordenador conectado en red actualiza y consulta las mismas bases de datos: la de precio de productos y la de existencias. Si los ordenadores no estuviesen conectados en red seria necesario dotar a cada equipo de las dos bases de datos y, además, al final de cada día, actualizar la base de datos de existencias para todos y cada uno de los equipos.
- Economía de la empresa: En una red tiene cabida cualquier equipo (un ordenador sin disco duro en la actualidad sirve para poco. Sin embargo, en una red puede utilizarse de forma que acceda al disco duro del servidor de la red).
- Comunicación entre usuarios: Dentro de una empresa, sin la presencia de una red, los trabajadores debían comunicarse por teléfono o desplazándose.
- Trabajo en grupo: Facilita el intercambio de información entre los distintos miembros de un grupo de forma rápida y cómoda.

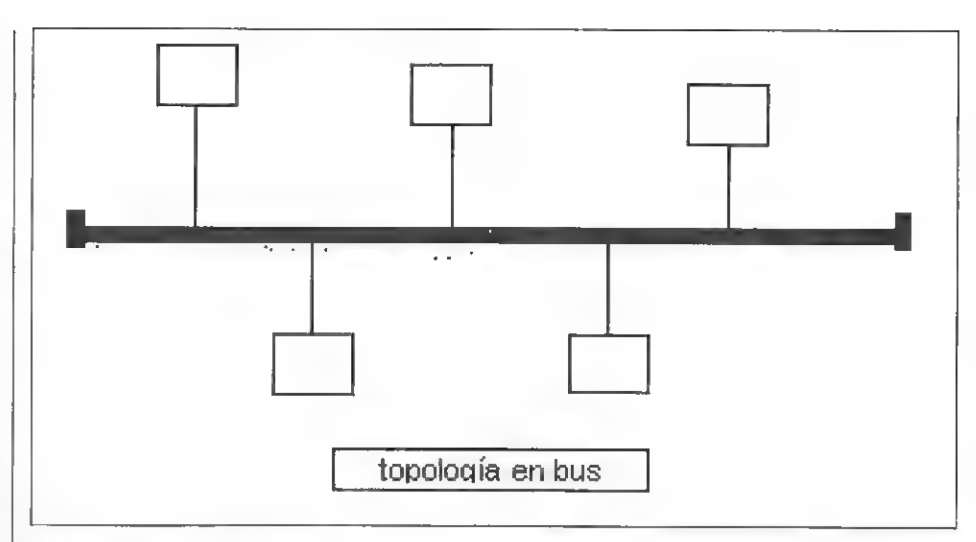


Figura 1. Topologías puras.

Las redes se catalogan de acuerdo al espacio físico por el que están distribuidas

Estos son los principales motivos por los que se plantea la necesidad de establecer una red, pero existen muchos más.

TOPOLOGÍA DE UNA RED

La topología de una red define únicamente la distribución del cable que interconecta los diferentes ordenadores. A la hora de instalar una red, es importante seleccionar la topología más adecuada a las necesidades, teniendo en cuenta factores como la distribución de los equipos a interconectar, tipo de aplicaciones que se van a ejecutar, inversión que se quiere hacer, coste que se quiere dedicar al mantenimiento y actualización de la red, tráfico que debe soportar la red, capacidad de expansión, entre otros. Las topologías puras son tres: *topología en bus*, *en estrella* y *en anillo* (ver figuras 1, 2 y 3). A partir de estas tres topologías se generan otras como son: *anillo-estrella*, *bus-estrella*, etc. Una breve introducción a cada una de estas topologías puede ser la siguiente:



Figura 3. Niveles del modelo OSI de la ISO.

- **topología en bus:** Se trata de un cable que pasa por todas las estaciones que conforman la red a forma de circuito serie (ver figura 1).
- **topología en estrella:** Consiste en un punto central (*nodo de cableado*) al que están conectadas todas las estaciones de la red, con cables independientes (ver figura 2).
- **topología en anillo:** Se trata de un cable que forma un círculo cerrado y al que están conectadas todas las estaciones (ver figura 3). En algunas ocasiones, la forma física de la red forma una estrella, aunque internamente se trata como un anillo. En este caso se dice que la red tiene topología de estrella pero se habla de anillo lógico.

ARQUITECTURA DE UNA RED

Cuando se habla de arquitectura de una red, no sólo se hace referencia a su topología, sino también al método de acceso al medio que la red utiliza. Por tanto el concepto de arquitectura de una red engloba al de topología, aunque habitualmente se confunden. En resumen, cuando se habla de arquitectura de una red, debe definirse su topología, así como el método que los nodos de la red utilizan para acceder al medio que conforma la red.

TERMINOLOGÍA BÁSICA.

Estación de trabajo: Cualquier ordenador conectado a la red. Antiguamente, sólo se llamaba estación de trabajo a los ordenadores más potentes, en la actualidad no es así. Esto no impide que la estación pueda trabajar de forma independiente y utilizar los servicios de la red cuando le sea necesario.

Nodo: Cualquier estación de trabajo, terminal, ordenador personal, impresora o cualquier otro dispositivo conectado a la red. Los dispositivos pueden conectarse a la red a través de un ordenador, o directamente, si son capaces de soportar una tarjeta de red.

Servidor: Se trata de una estación de trabajo que gestiona algún tipo de dispositivo de la red, como pueden ser impresoras, faxs, modems, discos, etc., dando servicio al resto de las estaciones. En función del servicio prestado se puede hablar de servidor de impresión, servidor de comunicaciones, servidor de ficheros, etc. Estos servidores pueden ser dedicados, cuando no pueden utilizarse para otra cosa, o no dedicados, cuando funcionan como cualquier otro ordenador, además de como servidor de algún elemento.

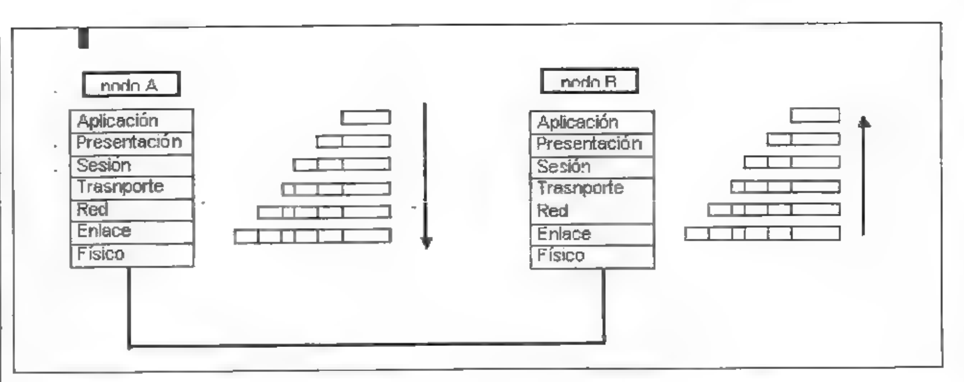


Figura 5. Comunicación entre nodos.

Medio de transmisión: Se trata de cualquier medio físico, que pueda transportar información en forma de señales electromagnéticas. El medio de transmisión es el soporte de toda red. Existen diferentes medios de transmisión: cable coaxial, fibra óptica, par trenzado, microondas, ondas de radio, infrarrojos, láser, etc.

Método de acceso al medio: Una vez que se tiene seleccionado el medio de transmisión que se va a utilizar para implementar la red, se debe elegir el método que los diferentes nodos de la red van a emplear para acceder a dicho medio. En principio, se podría obviar esta cuestión, pero si el lector se detiene un momento a pensar en el siguiente ejemplo, se dará cuenta de la necesidad de esta política. El ejemplo es el siguiente: Imagine que tiene dos ordenadores de su red local que quieren utilizar la red para enviar información en un instante determinado: si los dos ordenadores colocan en el medio físico, sin más, la información, puede ser que ambos paquetes de información "choquen" y se deterioren, no llegando ninguno de ellos a su destino. Obviamente, cuando varios dispositivos están compartiendo un medio común es necesaria la implantación de una política de uso de dicho medio: esto es el método de acceso al medio. En cada topología de red se utiliza el más conveniente de estos métodos: por ejemplo cuando se tiene una red en anillo, el método de acceso al medio utilizado es el *paso de testigo*, mientras que si se tiene una topología en bus, los *métodos de contención* son los más adecuados.

Protocolos de red: Ya se ha establecido cómo van a acceder los diferentes nodos a la red, ahora es necesario especificar cómo van a comunicarse entre sí. Los protocolos de red definen las diferentes reglas y normas que rigen el intercambio de información entre nodos de la red. Los protocolos establecen reglas a muchos niveles: desde cómo acceder al medio, hasta cómo encaminar información desde el origen hasta su destino, pasando por la descripción de las normas de funcionamiento de todos y cada uno de los niveles del modelo *OSI* de la *ISO*. Algunos

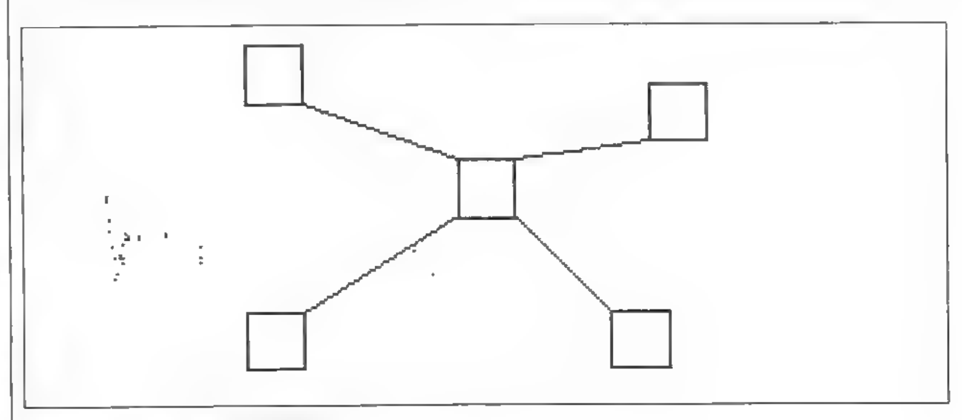


Figura 2: Topología en estrella.



ejemplos de protocolos, son: *TCP* (protocolo de control de transmisión), *IP* (protocolo Internet), *FTP* (protocolo para transferencia de ficheros), *X.25*, etc.

Sistemas operativos de red: Los sistemas operativos de red, además de incorporar las herramientas propias de un sistema operativo, como son, por ejemplo, herramientas para el manejo de ficheros y directorios, incluyen herramientas para el uso, gestión y mantenimiento de la red; herramientas destinadas a: correo electrónico, envío de mensajes, copia de ficheros entre nodos, ejecución de aplicaciones contenidas en otras máquinas, compartición de recursos hardware (impresoras, fax, etc), etc. Existen muchos sistemas operativos capaces de gestionar una red dependientes de las arquitecturas de las máquinas que se utilicen. Los más comunes son: Novell, Lantastic, Windows para trabajo en grupo, y Unix.

ISO (International Organization for Standardization): Se trata de una organización de normalización reconocida mundialmente. Su objetivo es el de promover y desarrollar normas para el intercambio internacional. Establece normas de estandarización en muchísimos campos, estableciendo modelos a seguir para todos y cada uno de ellos. Abarca campos tan dispares como el diámetro de algunos tipos de conectores, el paso de rosca de tornillos, el grosor de un modelo concreto de cable, etc. En cuanto al campo de las comunicaciones, la ISO ha desarrollado un modelo, al que llamó OSI.

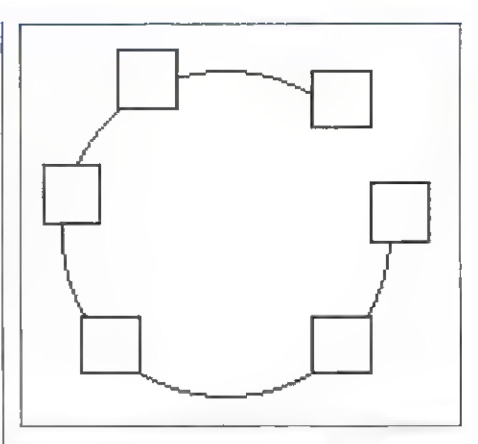
OSI (Open Systems Interconenection): Se trata de un modelo elaborado por la *ISO* que define los protocolos de comunicación en siete niveles diferentes. Estos niveles son: aplicación, presentación, sesión, transporte, red, enlace y físico. Cada nivel se encarga de una parte en el proceso de transmisión, (en el proceso de elaboración de la información a transmitir) apoyándose en los servicios que le ofrece el nivel inferior y dando servicios a niveles superiores. En la figura 4 pueden verse estos 7 niveles.

Paquete: Un paquete es básicamente el conjunto de información a transmitir entre dos nodos. Cuando una aplicación quiera enviar información a otra aplicación de otro nodo, lo que hace es empaquetar dicha información, añadiendo datos de control como la dirección de la máquina que envía la información (dirección origen) y la dirección de la máquina a la que va destinada la información (dirección destino). Por tanto, cuando se habla de empaquetamiento se hace referencia al proceso de guardar dentro de un paquete la información que se quiere transmitir.

Trama: El concepto de trama es de más bajo nivel que el de paquete. Una trama es un paquete o parte de un paquete (a veces los paquetes se fragmentan si son muy grandes) al que se le añade información de control. Las tramas son los elementos que circulan por el medio físico.

Dirección: Todos los nodos de la red deben tener una dirección que los identifique dentro de la red de forma única, al igual que todos tenemos una dirección postal para poder recibir correo. La dirección de un nodo depende del protocolo de comunicación que se está utilizando. Cada protocolo utiliza un sistema de direccionamiento diferente. Por ejemplo el protocolo IP (de la familia de protocolos TCP/IP) utiliza direcciones de 32 bits: los primeros identifican la red y los últimos identifican el nodo dentro de la red.

TCP/IP: Son dos protocolos de comunicaciones: el protocolo *TCP*(Protocolo de control de transmisión) que se establece a nivel de transporte del modelo *OSI* y el protocolo *IP* (Internet protocolo) que pertenece al nivel de red. En realidad cuando se utiliza el término *TCP/IP* se hace referencia a una familia muy amplia de protocolos representada por estos dos. Estos protocolos son los que utiliza Internet para la interconexión de nodos. Sobre ellos se establecen otros protocolos a niveles superiores hasta llegar al nivel de aplicación (el más cercano al usuario) en el que se encuentran protocolos tan conocidos como *FTP* (protocolo para transferencia de ficheros).


Figura 3 Topología en anillo.

La topología de una red define únicamente la distribución del cable que interconecta los diferentes ordenadores

Interconexión de redes: A veces, se plantea la necesidad de interconectar dos o más redes, por ejemplo, por la necesidad de compartir recursos o la división en dos subredes de una red, para mejorar el rendimiento de ésta. En ambos casos es necesaria la presencia de un dispositivo (que puede ser un *hubs*, un *bridges*, un *routers*, etc). Cada uno de estos dispositivos está disecado para interconectar redes. La diferencia estriba en el nivel en el que es necesario interconectarlas: no es lo mismo interconectar dos redes con la misma arquitectura, que dos redes de arquitecturas diferentes y con diferentes protocolos.

Hubs (concentradores): Se trata de un dispositivo que centraliza la conexión de los cables procedentes de las estaciones de trabajo. Existen dos tipos de concentradores: pasivos y activos. Los concentradores pasivos son simplemente cajas que disponen de unos puertos a los que se conectan las estaciones de trabajo dentro de una configuración en forma de estrella. Un concentrador activo es un concentrador que dispone de más puertos que un concentrador pasivo para la conexión de estaciones y que realiza más tareas, como puede ser la de amplificación de

la señal recibida antes de su retransmisión.

Bridges (puentes): Permiten dos cosas: primero, conectar dos o más redes entre sí, aún teniendo diferentes topologRas, pero asumiendo que utilizan el mismo protocolo de red. Segundo, segmentar una red en subredes. Los puentes trabajan en el nivel de enlace del modelo *OSI* de la *ISO*. Algunos de los motivos que pueden inducir a instalar un puente son: Ampliar la extensión de una red y/o el número de nodos que la componen, reducir el cuello de botella del tráfico causado por un número excesivo de nodos unidos, unir redes de topologías similares como bus y anillo. Los puentes se pueden crear incorporando dos tarjetas de red (una de cada una de las redes a interconectar) dentro del mismo servidor (conectado obviamente a ambas redes), siempre que el sistema operativo de red de dicho servidor sea capaz de gestionarlo. Existen dos tipos de puentes: locales y remotos. Los puentes locales sirven para segmentar una red y para interconectar redes que se encuentran en un espacio físico pequeño, mientras que los puentes remotos permiten interconectar redes lejanas.

Routers (encaminadores): Se trata de dispositivos que interconectan redes a nivel de red del modelo *OSI* de la *ISO*. Realizan funciones de control de tráfico, encaminamiento de paquetes por el camino más eficiente en cada momento.

Gateways (pasarelas): Se trata de ordenadores que trabajan a nivel de aplicación del modelo *OSI* de la *ISO*. Es el más potente de todos los dispositivos de interconexión de redes. Permiten interconectar redes de diferentes arquitecturas, es decir, de diferentes topologías y protocolos. Además, no sólo realiza funciones de encaminamiento, como los *routers*, sino que también realiza conversiones de protocolos, modificando el empaquetamiento de la información para adaptarla a cada red. En terminología Internet las palabras *router* y *gateway* se utilizan indistintamente.

CÓMO FUNCIONA UNA RED

Se han visto muchos conceptos relacionados con el mundo de las redes que se irán ampliando en próximos artículos, pero todavía no se ha explicado como funciona una red.

Un ejemplo práctico puede ser el correo. Cuando alguien desea mandar una carta a otra persona, la escribe, la mete en un sobre con el formato impuesto por correos, le pone un sello y la introduce en un buzón. La carta es recogida por un cartero, clasificada por personal de correos, según su destino, y enviada a través de medios de transporte hacia la ciudad destino. Una vez allí otro cartero irá a llevarla a la dirección indicada en el sobre. Si la dirección no existe, al cabo del tiempo la carta será devuelta por los mismos cauces que llego al supuesto destino.

La elección del medio de transmisión para una red, no se hace de forma aleatoria

Más o menos, esta es la forma en que funciona una red: la carta escrita es la información que se quiere transmitir. El sobre y el sello es el paquete con el formato impuesto por el protocolo que se utiliza en la transmisión. La dirección del destinatario es la dirección del nodo destino y la dirección del remitente, será la dirección del nodo origen. Los medios de transporte que llevan la carta cerca del destino es el medio de transmisión (cable coaxial, fibra óptica, aire,...). Las normas del servicio de correos, carteros, y demás personal, son los protocolos de comunicaciones establecidos. Supóngase que se está utilizando el modelo *OSI* de la *ISO*. Ya se ha dicho que este modelo tiene 7 niveles, es algo así como decir que la carta escrita pasa por 7 filtros diferentes (trabajadores con diferentes cargos) desde que se hecha al buzón hasta que llega al destino. Cada nivel de esta torre se encarga de realizar funciones diferentes en la información a transmitir.

Por ejemplo, una aplicación del nodo A (que obviamente se encontrará en el nivel de aplicación de la torre de niveles) quiere enviar información a una aplicación del nodo B (ver figura 5). La información a transmitir va descendiendo a través de la torre de protocolos del nodo A hasta llegar al último nivel (el físico) que es el que en realidad la transmite. Cuando llega al nodo B, la información ira ascendiendo por la torre de protocolos hasta llegar a la aplicación destino de dicha información.

Cada nivel por el que pasa la información a transmitir que se ha insertado en un paquete, añade información de control, que el mismo nivel en el nodo destino irá eliminando.

Cada nivel se encarga de cosas muy distintas: desde el control de errores, hasta la reorganización de la información transmitida cuando ésta se ha fragmentado en tramas. Más adelante se verá con más detalle el proceso que se realiza en cada nivel.

Si la información va dirigida a una red diferente (otra ciudad en el caso de la carta), la trama debe llegar a un dispositivo de interconexión de redes (*router*, *gateway*, *bridges*), que decidirá, dependiendo de su capacidad, el camino que debe seguir la trama. Por eso, es imprescindible que el paquete lleve la dirección destino y que ésta contenga, además de la dirección que identifica al nodo, la dirección que identifica la red a la que pertenece el nodo.

Ya se ha explicado parte del proceso de transmisión, pero aún falta contar cómo el destino se entera de que la trama que circula por el cable es para él. Existen muchas técnicas para esto, la más común consiste en que cada nodo que integre la red mire todas las tramas que circulan por ella, analizando la dirección destino de dicha trama para ver si se trata de información para él o no.

PRÓXIMO NÚMERO

En este articulo se han explicado los conceptos más relacionados y oídos del mundo de las redes. En el próximo número se profundizará en el tema de las distintas topologías de redes.

FICHEROS MIDI



Agustín Guillén

Comienza este mes una serie de artículos dedicada a estudiar los diferentes formatos de ficheros de sonido

Hasta hace bien poco, los ficheros que "solo" almacenaban notas y eventos, estaban reservados a unos pocos afortunados poseedores de tarjetas de sonido con bancos estándar de instrumentos grabados en sus ROMs, o con sintetizadores conectados a sus Pcs. La única posibilidad que tenían los demás aficionados de escucharlos estaba en los chips FM que incorporan casi todas las tarjetas de sonido, mediante la emulación de los bancos de instrumentos definidos en algún lugar del software. Pero con el aumento de prestaciones de las tarjetas musicales y con su bajada de precios, las tarjetas con potentes síntesis de sonido y con salida/entrada MIDI estándar están cada vez más extendidas.

INTRODUCCIÓN

Debido a las numerosas peticiones recibidas en la redacción, se comienza en este número una serie de artículo sobre formatos de ficheros de sonido. En ella van a tener cabida tanto los ficheros que almacenan sonidos en modo *raw* (WAV o VOC) como los que guardan eventos e información sobre notas musicales y canciones en general (MIDI), o los que mezclan ambos mundos (MOD, S3M, etc.), aunque estos últimos fueron descritos en la serie de artículos sobre sonido que actualmente se publicó en esta misma revista. Para comenzar se tratará el formato MIDI, su historia, su formato y sus conceptos.

DESCRIPCIÓN

El formato MIDI fue desarrollado por la compañía americana Opcode Systems y nació de la necesidad de intercambiar información y datos (canciones principalmente) entre los cada vez más extendidos programas secuenciadores bajo ordenadores personales. Las siglas MIDI significan *Musical Instruments Digital Interface* y vienen de la normas de interconexión entre instrumentos musicales.

En la figura 1 se describen los conectores MIDI y sus conexiones, como se observa, no podía ser más sencillo y se trata de una conexión serie que funciona a base de impulsos relativamente lentos. Su velocidad de transmisión es de 31.250 baudios con un margen de tolerancia del 1%, con lo que se obtiene (enviando 8 bits de datos, un bit de arranque y un bit de parada) una velocidad de 320 microsegundos para cada byte.

FORMATO

Los ficheros MIDI se componen de uno o más conjuntos de información completos. Cada conjunto contiene los datos sobre velocidad de reproducción, notas musicales, retardos entre cada nota y completas estructuras para cada pista musical. Físicamente estos conjuntos están formados a base de bloques o *chunks* y siempre se componen de un bloque de cabecera (*header chunk*) y de uno o más bloques de pistas (*track chunks*).

Cada bloque tiene una cabecera y unos datos. La cabecera se compone de un campo que define el tipo al que pertenece (*header* o *track*) y otro con la longitud de los datos que lo componen, esta longitud es solo de los datos y hay que sumarle 8 bytes, si se desea conocer el tamaño real del bloque completo.

Las cadenas ASCII de 4 caracteres que indican el tipo de bloque son: MThd (cabecera) o MTrk (pistas).
El esquema de un fichero MIDI quedaría de la siguiente forma:

Bloque de cabecera
Bloque de pistas
... ...

Repitiéndose este último bloque tantas veces como se quiera.

Bloque de cabecera
En él se especifican algunas informaciones básicas sobre el fichero:

Formato.- Indica la organización del fichero y tiene los siguiente valores:

0 = El fichero contiene una sola pista multi-canal

1 = El fichero contiene una o más pistas simultáneas de una secuencia.

2 = El fichero contiene uno o más patrones secuenciales de una sola pista.

Claramente el formato más estándar es el 0 (una sola pista multi-canal) y es el fichero más intercambiable, independientemente del software o de la máquina que lo interprete.

Número de bloques de pistas.- Informa del número de pistas incluidas en el fichero.

División.- Número de divisiones o particiones de un cuarto de nota representados por los *delta-times* (intervalos de tiempo indicados después de cada evento) en el fichero. Si el valor es negativo, entonces representa particiones de un segundo. Esto permite definir periodos de tiempo actuales y no solo periodos métricos.

En la actualidad la práctica totalidad de tarjetas de vídeo que se comercializan incorporan la norma VESA en sus BIOS

Meta-Evento	Decripción	Comentario
FF 00 02 *ssss*	Número de secuencia	Debe especificarse al principio de cada pista y antes de cualquier evento MIDI que se vaya a transmitir
FF 01 long text	Evento de texto	Almacena texto informativo de cualquier tipo
FF 02 len text	Mensaje de copyright	Debe contener los caracteres (C) y debe ser el primer evento de toda la pista
FF 03 len text	Nombre de secuencia o pista	Según el formato del fichero, será el nombre de la secuencia o de la pista
FF 04 len text	Nombre de instrumento	Literal con el tipo de instrumentación utilizada
FF 05 len text	Lírica o letras de la canción	Letras a ser cantadas durante la canción
FF 06 len text	Marca de texto	Un breve comentario del tipo: "Comienza el Coro"
FF 07 len text	Nota o comentario de una situación	Similar al anterior
FF 2F 00	Final de la pista	Evento obligatorio
FF 51 03 tttt	Configurar el tempo	Lo indica en microsegundos por cuarto de nota MIDI
FF 54 05 hr mn se fr ff	Tiempo de inicio de la pista	Indica el arranque de una determinada pista
FF 58 04 nn dd cc bb	Firma de tiempo	*nn* y *dd* son el numerador y el denominador de la firma. *cc* indica la velocidad del metrónomo. *bb* es el número de notas dentro de una nota MIDI.
FF 59 02 sf mi	Firma del teclado	Si *sf* es negativo indica teclas "flats", si positivo "sharps" y si 0 "tecla C". Si *mi* es 0 "tecla mayor" y si es 1 "tecla menor"
FF 7F len data	Meta-evento específico del secuenciador	Utilizado por cada fabricante para su propia información, indicando en los primeros bytes el identificador del fabricante

El formato del bloque de cabecera quedaría de la siguiente forma:

4 bytes MThd

4 bytes Longitud de los datos de la cabecera (6)

2 bytes Formato

2 bytes Número de bloques de pistas

2 bytes División

Bloque de pista
En el bloque de pista es en donde realmente se almacena la canción y se trata simplemente de una lista de eventos a ejecutar.

Se compone de una lista de eventos de pista y cada uno de ellos de un *delta-time* y del evento en si mismo:
Evento de pista
- *Delta-time*
- Evento

El *delta-time* representa la cantidad de tiempo antes de ejecutarse el siguiente evento. Puede utilizarse un *delta-time* de 0 si deben ejecutarse simultáneamente algunos eventos dentro de la pista. Los *delta-times* se almacenan en números de longitud variable, estos valores son representados como números de 7 bits por byte. Todos los

FIGURA 1

Conectores MIDI de 5 pins

MIDI In		MIDI Out	
Pin	**Descripción**	**Pin**	**Descripción**
1	N/C	1	N/C
2	Masa	2	Masa
3	N/C	3	N/C
4	Fuente	4	Sincronismo
5	Sincronismo	5	Fuente

bytes tienen el bit 7 activado, excepto el último, que lo tiene a 0. Para comprender mejor esto se lista a continuación unos ejemplos de números de longitud variable:

Numero original	Número representado
00000000	*00*
00000040	*40*
00000080	*81 00*
00002000	*C0 00*
001FFFFF	*FF FF 7F*
08000000	*C0 80 80 00*
0FFFFFFF	*FF FF FF 7F*

Como se observa el número mayor que se puede representar dentro de los 32 bits disponibles para el número de longitud variable es 0FFFFFFF. Aunque se podrían almacenar datos de un tamaño mayor, se considera este límite suficiente, pues incluso con un *tempo* rápido de 500 "tics" por segundo, obtendríamos ¡cuatro días entre eventos!.

Existen tres tipos de eventos:
Evento MIDI.- Se trata de cualquier mensaje dirigido a un canal MIDI

Meta-evento.- Siempre comienza con el valor FF y tiene el siguiente formato:

FF <tipo> <longitud> <bytes>

Especifica información no-MIDI utilizable por los secuenciadores o información útil para el propio formato MIDI. Todos los meta-eventos comienzan con el byte FF, despues continuan con el tipo (que será siempre menor de 128), luego viene la longitud de los datos contenida en un número de longitud variable y terminan con los datos propiamente dichos, cuya longitud ha sido previamente especificada

No es necesario que cada aplicación MIDI reconozca todos los meta-eventos existentes, pues cada fabricante creará y utilizará varios de manera exclusiva; con interpretar los más habituales, es suficiente.

En la figura 2 se encuentra una lista de los meta-eventos más comunes.

Evento exterior.- Van dirigidos a un determinado sistema MIDI, algo parecido a una secuencia Escape de las utilizadas por las impresoras. Dispone de dos formatos diferentes:
F0 <longitud> <bytes a transmitir después de F0>
F7 <longitud> <todos los bytes a ser transmitidos>

El formato del bloque de pista quedaría configurado de la siguiente manera:

4 bytes	MTrk
4 bytes	Longitud de los datos de la pista
¿¿¿???	Datos de la pista (eventos)

Si se desea más información sobre el formato o sobre las distintas especificaciones que lo componen, existe una asociación a nivel mundial que se encarga de dar soporte a todo lo referente a la normativa MIDI:

International Midi Association
5316 West 57th Street
Los Angeles, CA 90056
(415) 321-MIDI

EJEMPLO

A continuación se muestra en valores hexadecimales un completo fichero MIDI.

Bloque de cabecera:

4D 54 68 64	*MThd*
00 00 00 06	*Longitud del bloque*
00 00	*Formato 0*
00 01	*Una pista*
00 60	*Valor de 96 por cuarto de nota*

Bloque de pista:
Cabecera:

4D 54 72 6B	*MTrk*
00 00 00 3B	*Longitud del bloque (59)*

Eventos:

Delta-time	Evento	Comentario
00	*FF 58 04 04 02 18 08*	Firma de tiempo
00	*FF 51 03 07 A1 20*	*Tempo*
00	*C0 05*	
00	*C1 2E*	
00	*C2 46*	
00	*92 30 60*	
00	*3C 60*	
60	*91 43 40*	
60	*90 4C 20*	
81 40	*82 30 40*	*Delta-time de dos bytes*
00	*3C 40*	
00	*81 43 40*	
00	*80 4C 40*	
00	*FF 2F 00*	*Final de la pista*

Una parte complicada de la interpretación de los fichero MIDI es la lectura y escritura de los números de longitud variable, por lo que se incluyen las rutinas en C necesarias para su correcta manipulación.

```
EscribeVarLen (register long valor)
{
    register long buffer;
    buffer = valor & 0x7f;
    while ((valor >>= 7) > 0)
    {
        buffer <<= 8;
        buffer |= 0x80;
        buffer += (valor & 0x7f);
    }
    while (TRUE)
    {
        putc(buffer,outfile);
        if (buffer & 0x80)
            buffer >>= 8;
        else
            break;
    }
}

doubleword LeeVarLen ()
{
    register doubleword valor;
    register byte c;
    if ((valor = getc(infile)) & 0x80)
    {
        valor &= 0x7f;
        do
        {
            valor = (valor << 7) + ((c = getc(infile)) & 0x7f);
        } while (c & 0x80);
    }
    return (valor);
}
```



LOS CONTROLES

Juan Manuel y Luis Martín

En el artículo anterior se analizaron los formularios como el soporte básico para la realización de palicaciones con Visual Basic 4.0. En este artículo se analizarán los controles, que son los objetos gráficos que se sitúan sobre los formularios para que el usuario pueda interactuar con la aplicación.

Para la realización de aplicaciones Windows Visual Basic dispone de una serie de objetos gráficos que se sitúan en los formularios en la fase de diseño. Estos objetos gráficos se denominan controles. Ejemplos de controles son los botones de comando, los cuadros de texto, las listas, las casillas de verificación, etc.

Los controles, al igual que los formularios, poseen un repertorio de propiedades que definen tanto su aspecto como su comportamiento en tiempo de ejecución. Igualmente, disponen de un repertorio de procedimientos de evento que les permite responder a determinados eventos. Finalmente, cuentan también con una serie de métodos que permiten realizar operaciones con ellos.

ADICIÓN DE CONTROLES

Los controles se situan en los formularios como si fueran dibujados con un programa Windows de dibujo, utilizando para ello los movimientos y clics del ratón. El tipo de control que se desea añadir a un formulario se selecciona en la caja de herramientas. Para añadir un control a un formulario deben seguirse los siguientes pasos:

1. Hacer clic sobre el botón que representa el tipo de control dentro de la Caja de Herramientas, con lo que se mostrará pulsado.
2. Situar el puntero en forma de cruz en el punto del formulario donde se desea situar la esquina superior izquierda del control.
3. Pulsar el botón izquierdo del ratón y, manteniendolo pulsado, arrastrar el puntero hasta el punto donde se desea situar la esquina inferior derecha del control. Durante esta operación se visualiza un recuadro que representa al control, tal y como se muestra en la figura 1.
4. Liberar el botón del ratón, con lo que el control quedará situado en la posición indicada.

También es posible añadir un control haciendo un doble clic sobre el botón correpondiente de la Caja de Herramientas. En este caso, el control quedará situado en el centro del formulario con un tamaño predefinido, por lo que será necesario alterarlo posteriormente.

Si se desea añadir más controles del mismo tipo al formulario sin necesidad de volver a acceder a la Caja de Herramientas continuamente, será necesario mantener pulsada la tecla CONTROL durante toda la operación.

El nombre por defecto de los controles que se van añadiendo está formado por un texto indicativo del tipo de control, seguido de un número de orden (Command1, Command2, Text1, Text2, etc).

MANIPULACIÓN DE CONTROLES

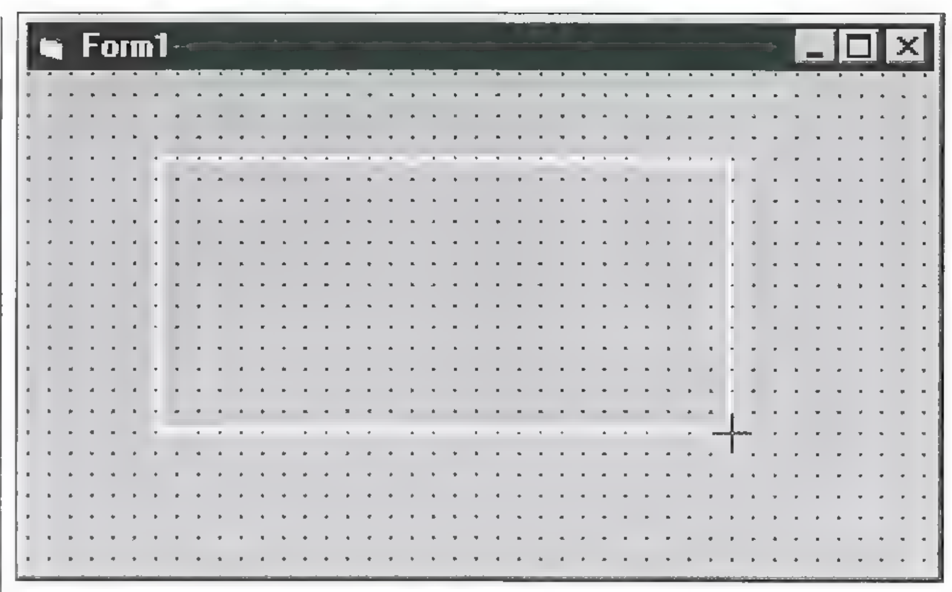
Para realizar cualquier operación de manipulación de un control en la fase de diseño, éste debe ser previamente seleccionado. Para ello, bastará con hacer clic sobre él. De esta forma, sobre el control se visualizan los controladores de tamaño, indicando que el control se encuentra seleccionado, tal y como puede verse en la figura 2.

Para cambiar la posición de un control dentro del formulario bastará con arrastrarlo hasta una nueva posición.

Durante esta operación se visualizará un recuadro representando la posición donde quedará situado el control.

Para cambiar el tamaño del control pueden segurise los siguientes pasos:

1. Seleccionar el control, haciendo clic sobre él.
2. Situar el puntero del ratón sobre uno de los controladores de tamaño, de forma que éste se visualizará como una doble flecha.
3. Pulsar el botón izqueirdo del ratón y, manteniendolo pulsado, arrastrar el borde del control en las direcciones indicadas por el puntero.
4. Liberar el botón del ratón, una vez alcanzado el tamaño deseado.



F igura 1: Recuadro de representación del control.

Es posible bloquear la posición de todos los controles dentro del formulario, de forma que estos no puedan ser movidos accidentalmente. Para ello, bastará con seleccionar la opción Bloquear controles del menú Edición, o hacer clic en el botón correspondiente de la Barra de Herramientas. Sin embargo, es posible mover los controles bloqueados manteniendo pulsada la tecla CONTROL durante la operación de arrastre.

Finalmente, para eliminar un control de un formulario, bastará con seleccionarlo y pulsar la tecla SUPRIMIR.

Visual Basic permite además utilizar el Portapapeles de Windows para la manipulación de los controles. De esta forma, es posble Cortar, Copiar y Pegar controles en un formulario. Para llevar a cabo las dos primeras operaciones será necesario seleccionar el control y la opción correspondiente del menú Editar. La opción Pegar sólo estará habilitada cuando se encuentre algún control en el portapapeles. Al pegarlo, éste quedará situado en la esquina superior izquierda del formulario. Debe tenerse en cuenta que con estas operaciones se copian las propiedades del control, pero no así el código de sus procedimientos de evento.

Tanto en tiempo de diseño como en tiempo de ejecución, la especificación de los valores de las propiedades de los controles se realiza de la misma forma que para los formularios. En tiempo de diseño, esta operación se realiza en la Ventana de Propiedades. Para ello, debe seleccionarse el control y acceder a la propiedad correspondiente dentro de la Ventana de Propiedades. Una vez seleccionada la propiedad, es posible teclear su valor o seleccionarlo en una lista, tal y como se vió en el artículo anterior.

En tiempo de ejecución, la asignación de valores a las propiedades de los controles se realiza mediante sentencias de asignación. La indicación de la propiedad se realiza indicando el formulario al que pertenece el control, el nombre del control y la propiedad, todo ello separado por puntos. La sintaxis general es la siguiente:

Formulario.Control.Propiedad = Valor

El nombre del formulario no es obligatorio, ya que por defecto se asume el formulario actual. S uso es recomendable pues puede evitar ambigüedades en el código, y ayuda a la claridad del mismo.

Los controles poseen un repertorio de propiedades que definen tanto su aspecto como su comportamiento en tiempo de ejecución

GRUPOS DE CONTROLES

Las operaciones explicadas hasta ahora para los controles pueden realizarse también sobre grupos de controles. De esta forma, es posible seleccionar varios controles, permitiendo moverlos, cortarlos, pergarlos, etc. todos a la vez. Del mismo modo, cuando se selecciona un grupo de controles, es posible asignar valores a las propiedades comunes a todos ellos.

Para seleccionar un grupo de controles deben seguirse los siguientes pasos:

1. Colocar el puntero del ratón en una zona libre del formulario que se encuentre por encima y a la izquierda del grupo de controles a seleccionar.
2. Pulsar el botón izquierdo del ratón y, manteniendolo pulsado, arrastrar el recuadro discontínuo que aparece hasta que éste englobe a todos los controles que se desea seleccionar. Hay que tener precaución para no incluir controles en la selección que no se desean agrupar.
3. Liberar el botón del ratón una vez englobados los controles a seleccionar.

Una vez liberado el botón del ratón, todos los controles que han sido selec-

cionados aparecen con los controladores de tamaño, tal y como puede observarse en la figura 3.

Existe otra posibilidad para realizar la selección de varios controles. Esta última consiste en hacer clic sobre cada uno de los controles que se desea seleccionar, pero manteniendo pulsada la tecla CONTROL. De esta forma, es posible seleccionar controles que se encuentren dispersos en el formulario y que no pueden englobarse en un área por la existencia de otros controles que no desea seleccionar.

Una vez selecconado un grupo de controles es posible moverlos todos a la vez por el formulario. Para ello, bastará con arrastrar cualquiera de los controles seleccionados. Durante esta operación aparecen una serie de recuadros que representan a todos los controles seleccionados, y el movimiento se realizará con todos los controles a la vez.

Al igual que sucede con los movimientos, también es posible realizar otras operaciones sobre el grupo de controles seleccionados. Entre ellas, es posible cortar, copiar y pegar todos los controles selecionados.

Como se ha mencionado anteriormente, otra de las operaciones posibles con un grupo de controles es asignar valores a las propiedades que sean comunes a todos ellos. Cuando se realiza la selección de un grupo de controles, la Ventana de Propiedades mostrará solamente aquellas propiedades que sean comunes a todos los controles selccionados.

Además, si se especifica algún valor dentro de esta ventana, éste se realizará de forma automática en la misma propiedad de todos los controles que se encuentran seleccionados. De esta forma, es posible ahorrar trabajo a la hora de indicar los mismos valores para varios controles.

PROPIEDADES COMUNES A LOS CONTROLES

La mayoria de controles de la Caja de Herramientas disponen de una serie de propiedades que son comunes a todos ellos. Algunas de estas propiedades son comnes tanto a controles como a formularios (Name, Caption, Font, BackColor, etc). El funcionamiento de estas propiedades es idéntico en ambos casos, y su significado fue descrito en el artículo anterior.

Los controles son objetos gráficos que se sitúan sobre los formularios para que el usuario pueda interactuar con la aplicación

Los controles también cuentan con una serie de propiedades que, aún siendo comunes a todos ellos, no lo son con las de los formularios, por lo que es preciso analizarlas. A continuación se analizarán una serie de propiedades que son comunes a la mayoría de controles.

Los controles disponen de una propiedad denominada Enabled que permite ctivar o desactivar un control. Cuando un control se encuentra desactivado, éste sigue visualizándose en el formulario, pero no puede responder a los eventos producidos por el usuario. Un valor True en esta propiedad permite habilitar el control, permitiendo a éste responder a los eventos. Por el contrario, un valor False deshabilita el control. Cuando el control se encuentra deshabilitado, además de no poder responder a los eventos, se visualiza en un color más pálido, para indicar al usuario la deshabilitación del mismo. La utilización de esta propiedad permite habilitar o deshabilitar controles en tiempo de ejecución, según lo requiera el estado de la aplicación.

Si, además de deshabilitar un control, se desea que éste no permanezca visible en el formualrio, es posible utilizar la propiedad Visible para ocultarlo. Un valor True en esta propiedad indica que el control será mostrado en el formulario, independientemente de que se encuentre habilitado o no. Por el contrario, un valor false hace que el control se oculte, por lo que el usuario no podrá ni visualizarlo ni interactuar con él.

EL ORDEN DE TABULACIÓN

Cuando se van añadiendo controles nuevos al formulario, Visual Basic les asigna un número de orden u Orden de Tabulación (TabOrder). Medinate este número se indica el orden en el que los controles irán capturando el foco según se vayan pulsando las teclas TAB y SHIFT+TAB. De esta forma, cuando en tiempo de ejecución, el usuario pulse la tecla TAB, el foco pasará al control que tenga el siguiente número de orden de tabulación. Si el usuario pulsa la tecla SHIFT+TAB, el foco pasará al control que tenga el número anterior de orden de tabulación.

El orden en que se reorrerán los controles se indica en la propiedad TabIndex de los mismos. Por defecto, Visual Basic asigna el valor a esta propiedad siguiendo el mismo orden de inclusión de controles en el formulario. Sin embargo, es posible alterar el valor

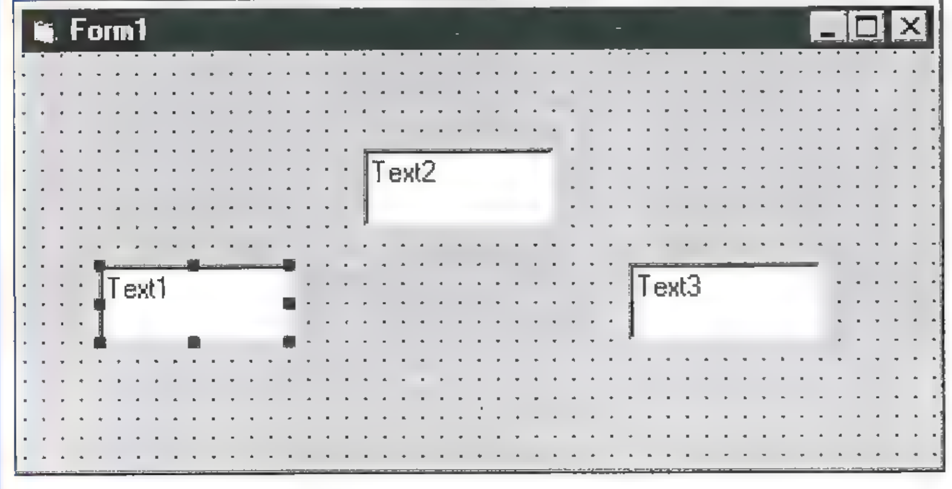


Figura 2: Selección de un control.

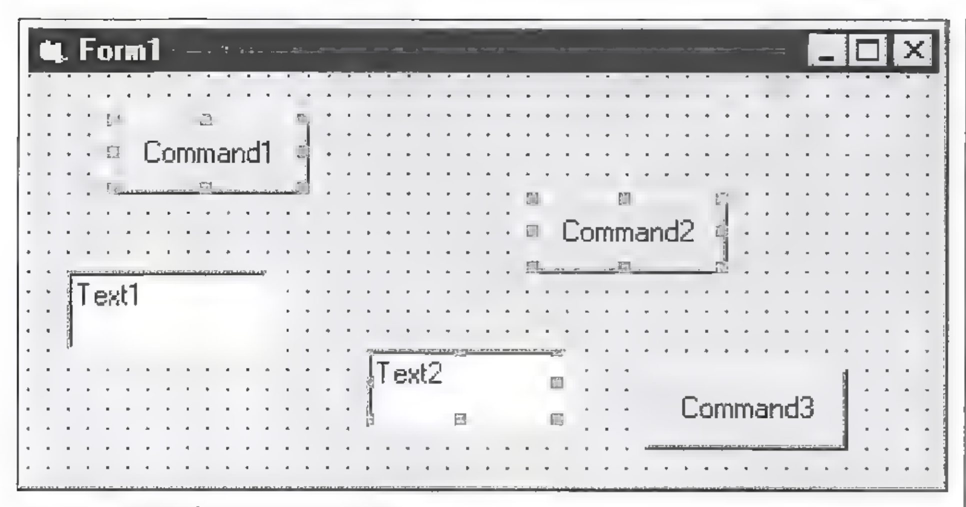


Figura 3: Selección de varios controles.

de dicha propiedad para así, alterar el orden de tabulación de los controles del formualrio. En tiempo de diseño, es posible conocer el orden de tabulación de los controles pulsando la tecla TAB. Mediante esta operación se irán seleccionando los distintos controles del formulario según el orden de tabulación establecido.

En determinadas ocasiones, es posible hacer que el foco no se detenga en alguno de los controles del formulario. Para ello, debe utilizarse la propiedad TabStop. Esta propiedad contiene un valor lógico que indica si el foco se detendrá o no en el control. Un valor True indica que el foco se detendrá en dicho control. Por el contrario, si se especifica un valor False, el foco no se detiene en el control, pasando automáticamente el foco al siguiente control en el orden de tabulación.

LA POSICIÓN

Uno de los aspectos importantes de los controles es el de la posición dentro del formulario. Al igual que en los formularios, las propiedades mediante las cuales es posible fijar la posición de un control son *Top* y *Left*. Su funcionamiento es igual al de los formularios, pero en esta ocasión, los límites los marca el formulario, es decir, estas propiedades definen la distancia entre el borde superior y el borde izquierdo del control y del formulario (no de la pantalla) respectivamente (figura 4).

Además, es posible incluir controles que no se encuentren situados por completo dentro del formulario. Cuando esto sucede, el formulario mostrará únicamente la parte que se encuentre dentro de su área de cliente. De cualquier forma, siempre será posible acceder a ellos, ya que Visual Basic no permite poner los controles fuera del área de cliente del todo, es decir, siempre tendrán una parte visible.

ASPECTO DE LOS CONTROLES

Otra propiedad de los controles es aquella que permite indicar el aspecto de los controles. Se trata de la propiedad Appearance y permite indicar si el control se visualizará con un aspecto plano o tridimensional. Un valor 0 (Flat) indica que el control se visualizará plano. Por el contrario, un valor 1 (3D) indica que el control se visualizará con aspecto tridimensional. La figura 5 muestra el aspecto de dos cuedros de texto con distintos valores en esta propiedad.

El resultado de esta propiedad depende a su vez del valzor de la propiedad Appearance del propio formulario. Además, cuando se asigna un valor 1 (3D) en algunos controles, su propiedad BackColor cambia, tomando como color de fondo el definido por Windows para sus botones. La forma de alterarlo es accediendo al Panel de Control del propio Windows.

TECLAS DE ACCESO RÁPIDO

Visual Basic permite acceder a los controles de un formulario de varias formas. Hasta ahora se han analizado dos: mediante un clic del ratón sobre el control, o mediante el orden de tabulación, es decir, pasando el foco de un control a otro hasta alcanzar el control deseado.

Además de estas dos formas, existe una tercera que, generalmente, proporciona un acceso más rápido que las anteriores. Se trata de la teclas de acceso rápido. De esta forma, el acceso a un control pude realizarse mediante la pulsación conjunta de la tecla rápida de dicho control junto con la tecla ALT.

Esta frma de acceso sólo se encuentra disponible para aquellos controles que posean la propiedad Caption. Para asignar una tecla de acceso rápido a un control, basta con acceder a su propiedad Caption y anteponer el carácter Ampersand (&) a alguna de las letras del texto. Cuando se realiza esta operación, la letra aparecerá subrayada. En tiempo de ejecución, el acceso al control puede realizarse pulsano conjuntamente la letra subrayada y la tecla ALT. Esta acción producirá el mismo efecto

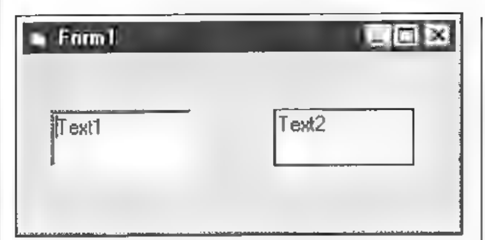


Figura 5: Diferencia de aspecto por la propiedad Appearance.

que si se realizase un clic sobre el control. Debe tenerse especial cuidado en no repetir la misma tecla rápida para dos controles distintos dentro del mismo formulario, pues, a pesar de no provocarse un error, el resultado de la pulsación de dicha tecla rápida es imprevisible.

EL PUNTERO DEL RATÓN

Al igual que sucedía con los formularios, Visual Basic permite cambiar el aspecto del puntero del ratón cuando éste se encuentra sobre algún control. De esta forma, es posible indicar al usuario alguna característica del propio control, o el estado del mismo.

Para especificar un icono distinto al utilizado en el formulario, puede utilizarse la propiedad *MousePointer*. Esta propiedad posee una lista de valores predeterminado, que coincide con la explicada en el artículo anterior sobre formularios. Si se especifica un valor en esta propiedad, cada vez que el puntero del ratón se encuentre sobre el control, el aspecto del mismo cambiará al indicado en la propiedad.

También es posible utilizar punteros personalizados. En este caso, será necesario indicar un valor 99 (*vbCustom*) en la propiedad *MousePointer*. De esta forma, se indica que el puntero a utilizar es un puntero personalizado. Además, el puntero debe especificarse en la propiedad *MouseIcon*, en la cual puede especificarse un archivo de icono (.ICO) o de cursor (.CUR). Si la especificación de esta propiedad se realiza en tiempo de diseño, es posible indicar el archivo mediante un cuadro de diálogo. Para ello, basta con pulsar el botón con los tres puntos suspensivos que aparece en la propiedad, dentro de la Ventana de Propiedades. Si, por el contrario, esta propiedad se indica en tiempo de ejecución, es decir, dentro del código,

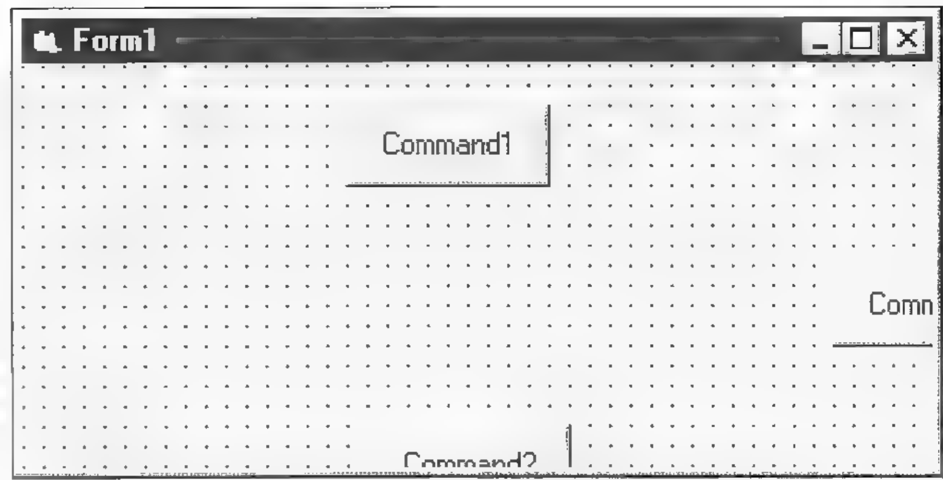


Figura 4. Aspecto del formulario con controles fuera del área de cliente.

será necesario utilizar la función *LoadPicture* para poder cargar un nuevo puntero.

Command1.MouseIcon = LoadPicture("Archivo")

Aunque las versiones de 32 bits de Windows (Windows'95 y Windows NT) permiten utilizar cursores animados (cursores cuyo aspecto puede ir variando con el tiempo, almacenados en archivos .ANI), Visual Basic no permite la utilización de este tipo de cursores en las aplicaciones desarrolladas con él.

Visual Basic proporciona además una extensa librería de iconos que es posible utilizar dentro de cualquier aplicación. La librería de iconos se encuentra en el subdirectorio *Icons* del directorio de Visual Basic. Dentro de este subdirectorio aparecen un grupo de directorios, cada uno de los cuáles contiene un numeroso grupo de archivos de icono, organizados por temas. Además, si se desea consultar los iconos, el manual del programador incluido en el paquete contiene un apéndice en el que aparecen todos los iconos con el nombre del archivo en el que se encuentran almacenados.

OTRAS PROPIEDADES

Al igual que sucedía con las propiedades de los formularios, existen muchas otras propieades dentro de los controles. Algunas de ellas son propias de cada control, mientras que otras son comunes a todos o parte de ellos. Esto es lo que sucede con las vistas hasta el momento.

Las propiedades propias de cada control se irán analizando al tiempo en que se analicen los diversos controles en los sucesivos artículos. Por otro lado, el resto de propiedades comunes corresponden a temas concretos más avanzados y se analizarán en los artículos correspondientes a dichos temas.

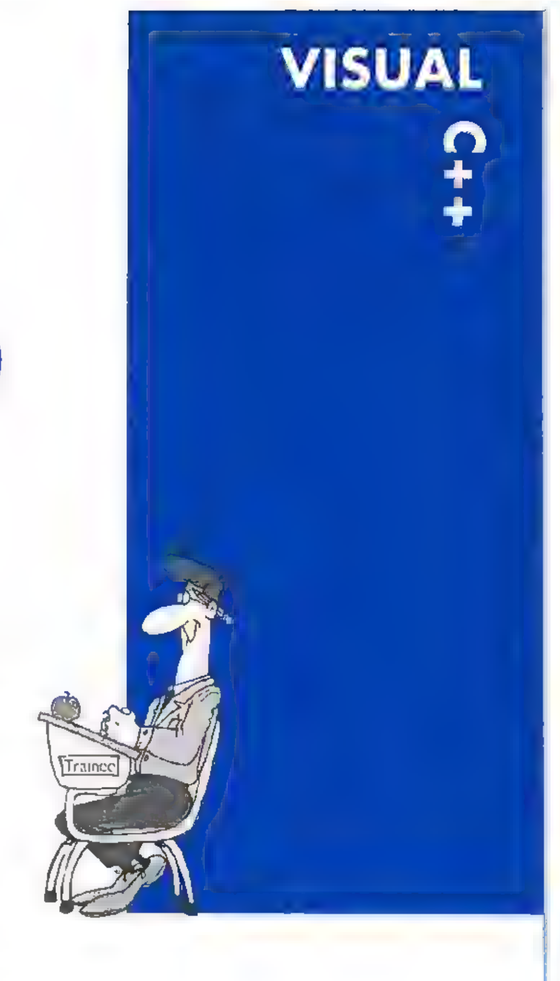
De entre las propiedades particulares de los distintos controles cabe destacar la propiedad Interval del temporizador (que permite indicar el inetrvalo de tiempo entre eventos), las propiedades de salto de las barras de desplazmiento, o las de ruta de los controles de lista de directorios y archivos.

Dentro de las propiedades comunes a los controles caben destacar las propiedades relacionadas con el tema de la ayuda en línea como HelpContextID, o las propiedades de enlace a datos como DataSource.

CONCLUSIÓN

En el presente artículo se han presentado las operaciones básicas que pueden realizarse con controles. Además se ha introducido una primera explicación de las propiedades más básicas de los controles, especialmente aquellas que son comunes a la mayoría de los controles. Es conveniente que el lector practique la utilización de estas propiedades realizando la inclusión de controles y alterardo tanto su posición como su tamaño, así como el valor de algunas propiedades para comprobar personalmente el resultado.

PROGRAMACIÓN MEDIANTE MFC (II)



Jorge del Río

Después de crear lo que se conoce como el esqueleto de la aplicación, deben añadirse todas las opciones y acciones que se hayan diseñado, mediante la utilización del editor de recursos y enlazar el código fuente con otra de las herramientas de la programación visual, ClassWizard.

La segunda etapa en el proceso de creación de una aplicación para Windows utilizando Visual C++ 4.0, es la adaptación e inclusión de nuevos piezas de lo que forma el interfáz de usuario. En esta parte se deben crear las ventanas, iconos, bitmaps, diálogos,... para definir lo que será la forma de interactuar de los usuarios con la aplicación. Si bien el interfáz de la aplicación no crítico para el funcionamiento de la misma, sí se sabe que es justamente éste el que mueve a unos usuarios a decidir si es ese el programa que buscan. Por ello, y siempre dentro de posibilidades del lector, debe crearse un interfáz agradable, sencillo de utilizar y que permita sacarle el mayor rendimiento al núcleo de la aplicación.

Para definir el interfáz de usuario, Visual C++ 4.0 ofrece una herramienta ya conocida por los programadores de versiones anteriores, pero que ha evolucionado de modo notable, permitiendo crear interfaces más profesionales con menor gasto de tiempo. Es el editor de recursos, la herramienta que permite definir de forma gráfica o visual todos las partes de la interfáz de la aplicación. Este editor permite crear el conocido archivo de recursos, con extensión RC, que contiene todas las sentencias que permiten al compilador crear los recursos y enlazarlos con la aplicación final.

En las aplicaciones Windows, los recursos se enlazan en último lugar y sobre un segmento especial denominado segmento de recursos, que és un segmento de datos de sólo lectura. Este segmento especial contiene todos los bitmaps, iconos, menús, diálogos, etc. que luego serán cargados y utilizados por la aplicación. Además, al estar aislados, el mismo editor de recursos permite visualizar y modificar directamente los recursos de un fichero ejecutable, sin por ello tener que recompilar o enlazar.

EL EDITOR DE RECURSOS

Además de las muchas novedades y mejoras introducidas por el editor de recursos, cabe destacar la total integración de dicho editor dentro del entorno de desarrollo integrado. Como ya se ha mencionado anteriormente, el editor de recursos permite crear y trabajar con todos los recursos de la aplicación como son:

- Cursores, que es la imagen del apuntador del ratón.
- Iconos, que son imágenes que suelen dar información gráfica.
- Bitmaps, que son mapas de bits o imágenes.
- Barra de herramientas, que son una serie de botones que permiten ejecutar determinadas acciones.
- Cajas de Diálogo, que son ventanas que permiten interactuar con los usuarios para que elijan entre determinadas opciones.
- Menús, donde se incluyen acciones a ejecutar por la aplicación.
- Aceleradores, son combinaciones de teclas que permiten ejecutar opciones de los menús.
- Tablas de cadenas de caracteres, que son cadenas de caracteres para ser visualizadas.
- Control de versiones, que tiene una serie de campos donde se pueden insertar la información sobre el programa, autores, empresa, etc.

Además del editor propiamente

dicho, existe una ventana que nos muestra una visión jerarquizada en función del tipo del recurso, de todos los existentes en la aplicación. Desde esta ventana se pueden añadir nuevos recursos, importarlos, eliminarlos o ver sus propiedades, simplificando el control y mantenimiento de los mismos.

Dentro del editor de recursos existen diferentes herramientas para crear, del modo más sencillo, cada tipo de recursos. En las siguientes líneas se van a tratar cada una de ellas. (Ver Figura A).

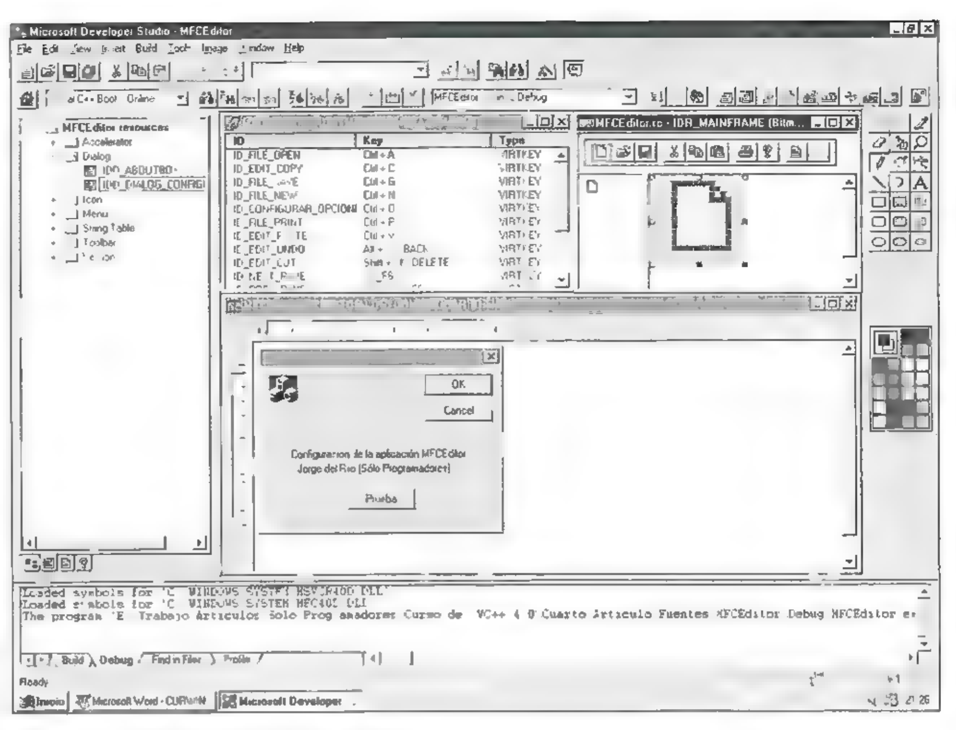


Figura A. Entorno de edición de recursos.

EDITOR DE CURSORES, ICONOS Y BITMAPS

El editor de cursores, iconos y bitmaps es básicamente el mismo. Para crear y modificar los cursores, iconos y bitmaps se han incluido nuevas herramientas dignas de un buen editor de gráficos y que permitirán diseñar imágenes de gran calidad. Además, en el caso de los cursores, se pueden elegir plantillas cuando se crean, lo que permite no partir de cero en el diseño del cursor.

EDITOR DE MENÚS Y ACELERADORES

Para crear y modificar los menús, existe una herramienta que permite diseñar y verificar visualmente los cambios realizados. Además, permite ver los menús del modo normal y como un popup o menú flotante. Para añadir una nueva opción o popup sólo es necesario escribir sobre las cajas de texto que se observan en la zona más a la derecha en el caso de los popups y abajo en las opciones.

También se incluye una utilidad para crear los aceleradores del menú. Mediante un cuadro de diálogo es posible pulsar las teclas deseadas y, automáticamente, se creará el acelerador.

La segunda etapa en el proceso de creación de una aplicación es la adaptación e inclusión de nuevas piezas que de forma a el interfaz de usuario

EDITOR DE LA BARRA DE HERRAMIENTAS

Esta utilidad permite, también de forma visual, crear los botones de la barra de herramientas, desde el icono del botón hasta su identificador y colocación. Esta utilidad será ampliamente utilizada ya que en todas las aplicaciones para Windows se tiende a añadir las barra de herramientas.

CONTROL DE VERSIONES

Mediante esta utilidad es posible incluir información interesante al ejecutable, que permitirá conocer la versión, autor, fabricante, etc... La utilización es sencilla, ya que sólo se deben de rellenar una serie de campos. Debe tenerse la precaución de mantener actualizada esta información con el objetivo de no sacar varias versiones del ejecutable con los mismos datos de la versión.

EDITOR DE CAJAS DE DIÁLOGO

El editor de cajas de diálogo permite diseñar este tipo de ventanas, así como añadir todos los controles que puedan ser necesarios. Tiene varios botones que permiten realizar diferentes ajustes para la colocación de los diversos controles.

ENLAZADO DE LOS RECURSOS CON EL CÓDIGO C++

Una vez que los diferentes recursos de la aplicación han sido creados o parcialmente creados, es posible darles funcionalidad, para lo que se necesitará enlazar dichos recursos con el código C++. La herramienta que se encarga de facilitar esta labor es ClassWizard.

ClassWizard permite añadir nuevas clases a una aplicación, gestionar las ya existentes, así como sus funciones miembros. Además, permite automatizar el proceso de enlazado de los recursos y el código C++.

Para ilustrar el método de enlazado de recursos con el código se va a plantear el siguiente ejemplo. Se desea añadir una nueva desplegable al menú del editor MFC llamado CONFIGURAR, y dentro de este una opción llamada OPCIONES. Además, se desea





Figura B. El botón para abrir ClassWizard.

añadir un botón en la barra de herramientas que tenga el mismo efecto que la acción del menú, y que despliegue una caja de diálogo. También se quiere que esto se ejecute cuando el usuario pulse las teclas CTRL+O, por lo que también habrá que crear un acelerador.

La primera acción es añadir al menú del fichero de recursos este nueva opción, indicando como identificador *ID_CONFIGURAR_OPCIONES*, como caption se va a indicar "Opciones\ tCTRL+O", es decir, el título que se desea ponerle, un tabulado y el acelerador que se le va a asignar. Esto último sólo es informativo y la asignación del tabulador debe realizarse en el editor de aceleradores. Además, se va a incluir el texto "Abre el diálogo de opciones" en el campo prompt. Este texto será visualizado en la barra de estado al tener el ratón situado encima de la opción del menú.

En un segundo paso se va a crear el acelerador, por lo que se abre la tabla de aceleradores del fichero de recursos. Haciendo un doble click en la última línea se abre un cuadro de diálogo que permite añadir un nuevo acelerador. El campo *ID* debe volver a ser rellenardo con *ID_CONFIGURAR_OPCIONES* y para determinar la combinación de tecla habrá que pulsar sobre el botón *NEXT KEY TYPED*, que va a permitir pulsar la combinación de teclas que se desea establecer como acelerador. Como se quiere que sea CTRL+O, debe pulsarse esta combinación y automáticamente se establece.

El siguiente paso a tomar es la inclusión del nuevo botón dentro de la barra de herramientas, por lo que se abre dicha barra desde el editor de recursos.

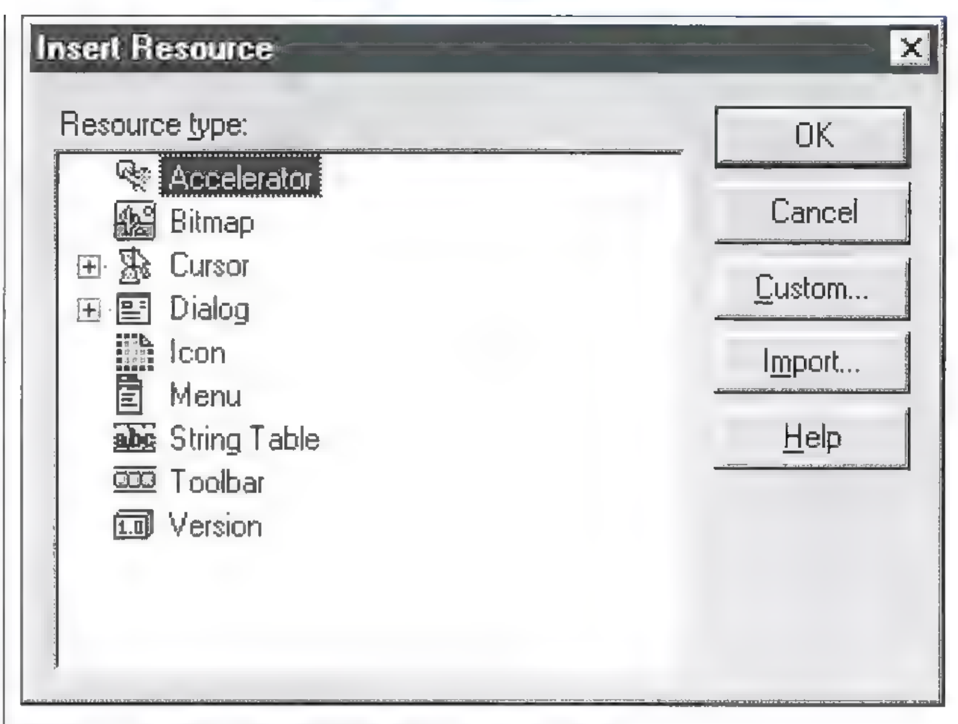


Figura C. Ventana de elección del tipo de recursos a crear.

Para añadir un nuevo botón debe hacerse un doble click sobre el botón libre que siempre está más a la derecha. Al abrirse la caja de diálogo de configuración hay que introducir en el campo *ID*, de nuevo, el identificador *ID_CONFIGURAR_OPCIONES*. Después de esto, debe diseñarse el dibujo que llevará el botón en su interior.

Una vez realizados los tres pasos anteriores, debe crearse el enlace con el código fuente, y para ello habrá que situarse en el editor del menú y seleccionar la opción OPCIONES, pulsar el botón de ClassWizard (Ver figura B). Inmediatamente surge la ventana de ClassWizard y más concretamente la sección de mapa de mensajes, donde se configuran las funciones que van a atender a los mensajes de Windows. En nuestro caso nos aparecerá en la sección de Object IDs el identificador ID_CONFIGURAR_OPCIONES y en la sección Messages los dos mensajes posibles que podemos atender, COMMAND y UPDATE_COMMAND_UI. En nuestro ejemplo vamos a atender al mensaje COMMAND, por lo que lo seleccionaremos con un click y pulsaremos sobre el botón Add Function. Tras este paso nos saldrá una ventana para indicar el nombre de la función, tomando el nombre por defecto. Sólo falta pulsar el botón Edit Code, lo que nos permitirá editar el código que será ejecutado al pulsar la opción del menú, pulsar las teclas CTRL+O o pulsar el botón de la barra de herramientas, ya que todos ellos tienen el mismo identificador.

ClassWizard permite añadir nuevas clases a nuestra aplicación, gestionar las ya existentes, así como sus funciones miembros

Para realizar la primera prueba vamos a incluir la siguiente línea de código:

MessageBox("Pulsado OPCIONES");

Al ejecutar el programa debemos verificar que dicho mensaje es mostrado al pulsar la opción del menú, de la barra de herramientas o CTRL+O. Además también debemos verificar que al pasar el ratón por encima del botón o de la opción del menú, en la barra de estado se cambia el texto poniendo "Abre el diálogo de opciones".

Añadir nuevas clases a la aplicación

Como ya habíamos indicado anteriormente en el ejemplo, cuando pulsa-

mos el botón OPCIONES, se debe visualizar una caja de diálogo, y en la parte anterior sólo mostrábamos un mensaje. En este apartado vamos a detallar el método para añadir una nueva caja de diálogo y por tanto una nueva clase que gestione sus acciones.

Para crear una nueva caja de diálogo, sólo hay que pulsar CTRL+R y seleccionar el tipo diálogo en la lista de recursos (Ver figura C). Además de indicar un diálogo podemos elegir de una lista de diálogos predefinidos, entre los que se encuentran los necesarios para crear una barra de herramientas o las cajas de propiedades de los controles OCX.

En este ejemplo hemos utilizado un diálogo estándar añadiendo un icono, texto y un botón que nos va ha permitir ilustrar como se atrapan los mensajes de los controles incluidos dentro de un diálogo. Para crear la clase C++ que gestione las acciones sobre el diálogo que acabamos de crear, debemos seleccionar el diálogo y pulsar el botón mágico, es decir, el botón de ClassWizard.

Como este diálogo no tiene asignada ninguna clase, nos aparece una ventana para indicarnos que el diálogo seleccionado no tiene efectivamente ninguna clase asociada (Ver figura D) y nos pregunta si deseamos crear una nueva clase, importar una clase de otro proyecto o bien seleccionar una de las ya existentes en este proyecto y utilizadas en otros diálogos. En nuestro caso y como queremos crear una clase nueva, debemos indicar esta opción, con lo que nos aparece una nueva ventana para que insertemos los datos de la nueva clase. En esta ventana podemos indicar el nombre asignado a la clase, seleccionar el nombre del fichero fuente que deseamos crear para incluir la clase, el nombre del fichero de cabecera donde se incluirá la definición de dicha clase, y que debe ser incluido en todos los módulos que utilicen este tipo de diálogo. Además, debemos indicar la clase madre de la cual deriva nuestra clase, siendo por defecto CDialog. Esta opción también será válida en nuestro caso. Otra opción a seleccionar es la posibilidad de que nuestra clase pueda ser utilizado por los clientes de la automatización OLE. Como último punto de decisión, debemos indicar si deseamos que la clase creada se incluya al Component Galery, es decir, a la galería de clases existentes y que pueden ser sencillamente en otros proyectos.

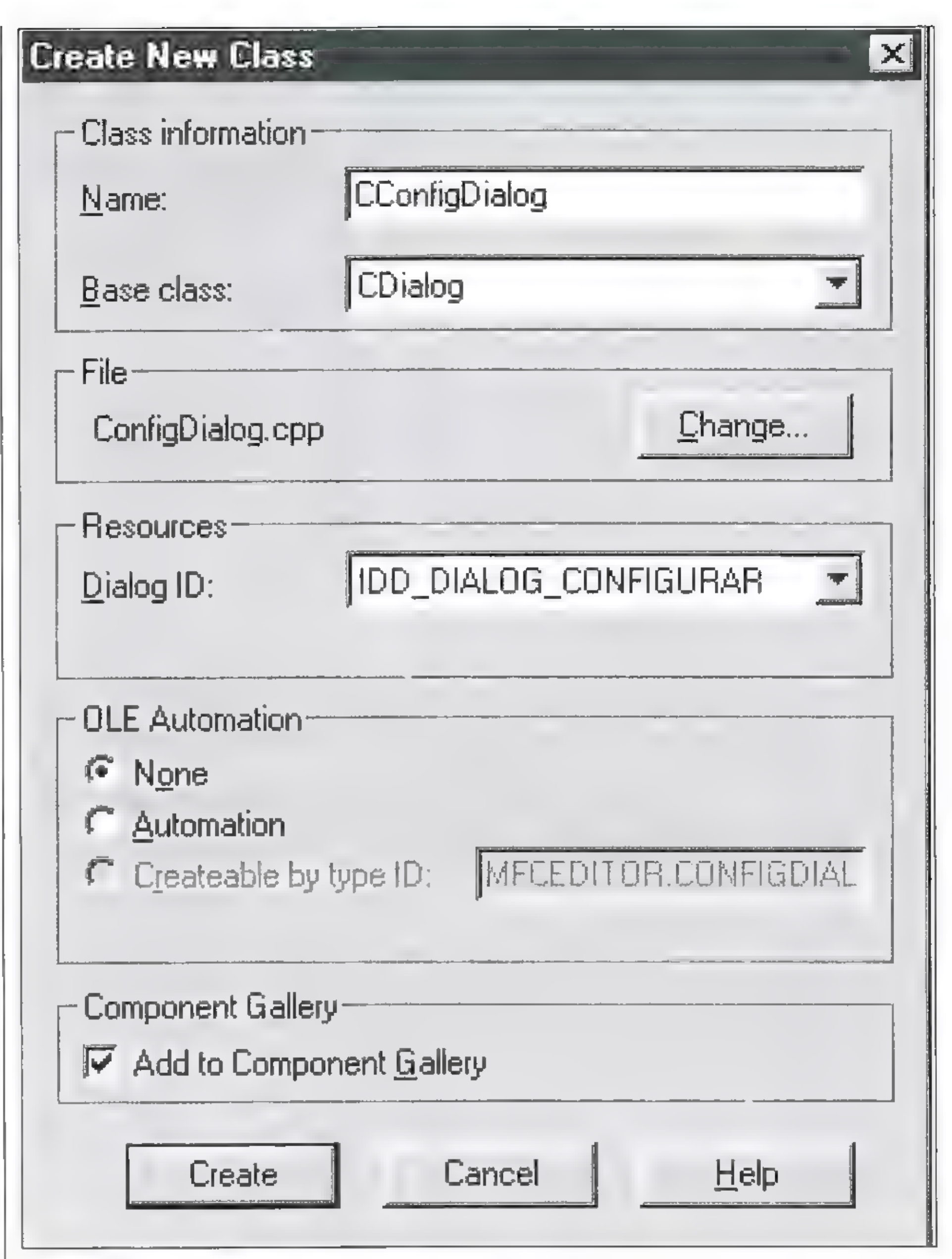


Figura D. Ventana de para crear nuevas clases.

Cualquier clase creada con la aplicación puede ser utilizada por los clientes de la automatización OLE

Una vez terminado el proceso anterior, de nuevo y con el fichero y la clase ya creadas, la ventana de mensajes estará en primer plano lo que nos permitirá seleccionar los mensajes que deseemos atender.

El proceso es sencillo, intuitivo y nos libera de una pesada tarea como es definir la clase, crear los ficheros, etc..., automatizando de nuevo la parte pesada y que no aporta nada al programa, disminuyendo el tiempo en la creación y sobre todo librándonos de más de un error fruto de teclear mal algún nombre de variable, palabra clave,...

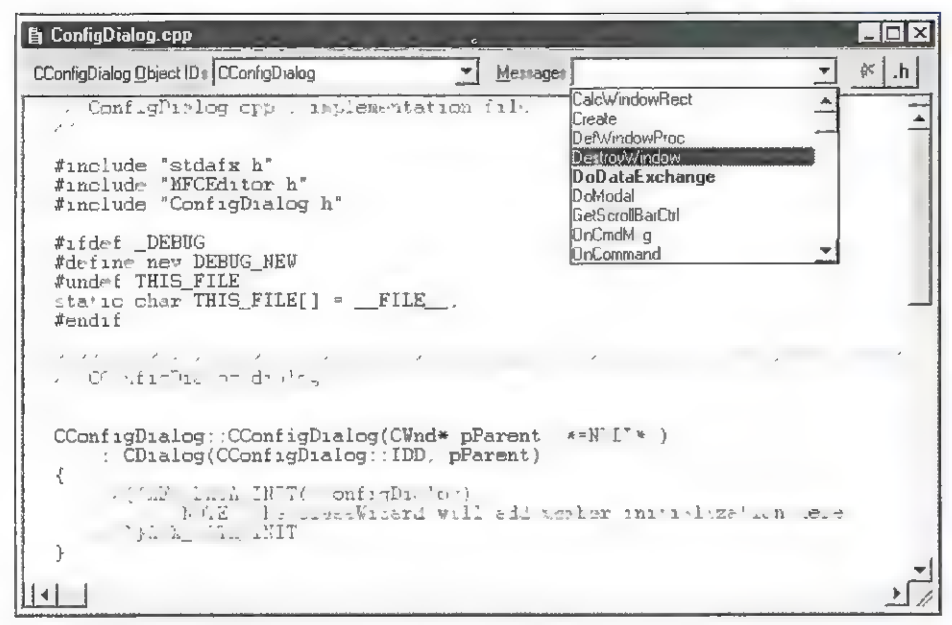


Figura E. Ventana de código fuente con ClassBar.

ATENDER LAS ACCIONES DEL DIÁLOGO CREADO

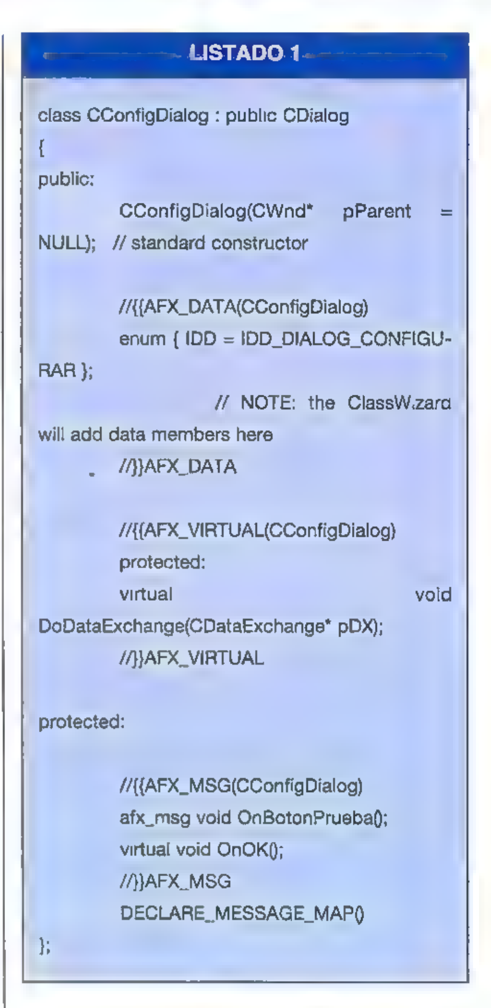
En la misma línea que el resto de las acciones, el trabajo de atender las posibles acciones dentro de cada uno de los diversos controles del diálogo es muy sencillo. Para ello sólo es necesario seleccionar el control que queremos atender y pulsar de nuevo el botón del ClassWizard. Al pulsar dicho botón se abre la ventana de ClassWizard, mostrándonos en el campo OBJECT Ids el identificador del control seleccionado, en nuestro ejemplo le llamamos ID_BOTON_PRUEBA, y en el campo de mensajes se muestran los dos mensajes que pueden ser atendidos en un botón. Estos mensajes son BN_CLICKED y BN_DOUBLECLICKED. Al seleccionar uno y pulsar el botón de añadir, se crea el código preciso para su utilización. Los conocedores de la programación en C del API de Windows sabedores de los procesos necesarios anteriormente, valorarán en mayor medida todos estos detalles de automatización.

Para crear una nueva caja de diálogo, sólo hay que pulsar CTRL+R y seleccionar el tipo *diálogo* en la lista de recursos

UTILIZACIÓN DE CLASSBAR

Una de las novedades de la versión 4.0 de Visual C++ es la inclusión de la herramienta ClassBar (Ver Figura E). Esta herramienta es una barra colocada encima de los ficheros fuente que nos permite crear el código necesario para atender a las funciones miembros de las diferentes clases. A los programadores con experiencia en Visual Basic les resultará muy familiar, porque es una adaptación de la que se utiliza en este lenguaje cuando se edita código. La utilización de ClassBar nos permite mejorar más aún si cabe el proceso de atención de mensajes de cada ventana.

Para ilustrar el manejo de esta herramienta, vamos a abrir el fichero fuente de la clase ConfigDialog y vamos a atender al mensaje producido cuando se pulsa la el botón OK, haciendo sonar un pitido. En la lista de identificador de objetos seleccionamos IDOK (identificador del botón en cuestión) y en la lista de mensajes seleccionamos BN_CLICKED. Al seleccionar este mensaje, se muestra una ventana para pedirnos el nombre de la función, que en nuestro caso vamos a dejar el que Visual C++ nos ofrece por defecto, es decir, OnOK. Al pulsar aceptar, se puede observar como se ha incluido la función que va a atender al mensaje, y en donde insertaremos la función MessageBeep(-1) para emitir el pitido.


LISTADO 1

```
class CConfigDialog : public CDialog
{
public:
        CConfigDialog(CWnd* pParent =
NULL);  // standard constructor

        //{{AFX_DATA(CConfigDialog)
        enum { IDD = IDD_DIALOG_CONFIGU-
RAR };
                // NOTE: the ClassWizard
will add data members here
        //}}AFX_DATA

        //{{AFX_VIRTUAL(CConfigDialog)
        protected:
        virtual void
DoDataExchange(CDataExchange* pDX);
        //}}AFX_VIRTUAL

protected:

        //{{AFX_MSG(CConfigDialog)
        afx_msg void OnBotonPrueba();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

EL CÓDIGO FUENTE GENERADO

Hasta ahora hemos ido avanzando dejando de un lado el código generado y observando como casi por arte de magia el resultado era el esperado, pero ¿Qué código hemos generado?

La contestación a esta pregunta pasa por analizar los dos ficheros creados, el fichero fuente ConfigDialog.cpp y el fichero de cabecera ConfigDialog.h.

El primer fichero a examinar es el de cabecera y tiene el aspecto que muestra el Listado 1. (Se han suprimido los comentarios).

Como se puede observar en el fichero de cabecera se incluye la definición de la clase, la cual deriva de CDialog como habíamos indicado. Además Visual C++ ha creado un constructor por defecto y la función DoDataExchange para que la sobrecarguemos e introduzcamos nuestro código. Esta función está entre dos líneas de comentario especiales (AFX_VIRTUAL), que delimitan la zona de funciones virtuales que Visual C++ puede modificar y que no deberíamos editar directamente. Lo mismo ocurre en la secciones AFX_DATA utilizado para las transferencias de datos en los controles y AFX_MSG utilizado por Visual C++ para incluir los prototipos de las funciones que atienden a los mensajes de la ventana. En nuestro caso se pueden visualizar las dos funciones creadas, OnBotonPrueba y OnOK. También hay que destacar en la zona de AFX_DATA como IDD se inicializa con el valor del identificador del diálogo. Como se verá en el constructor IDD será utilizado al llamar al constructor de la clase padre, lo que permitirá enlazar la clase ConfigDialog con su correspondiente diálogo.

En el fichero fuente se incluye el código mostrado por el Listado 2. (Se han eliminado los comentarios y líneas sin interés).

Como se puede ver están los cuerpos de las funciones miembros definidas. La primera es el constructor, que por defecto no tiene función, pero que contiene una zona a reseñar. Dicha zona es la delimitada por las líneas de comentario AFX_DATA_INIT, que nos indica que Visual C++ puede insertar en ese espacio la inicialización de los controles del diálogo. La siguiente función es DoDataExchange, y es una función llamada cuando los datos deben actualizarse con los valores de los controles. (En capítulos posteriores entraremos en detalle sobre estas zonas especiales).

La utilización de ClassBar nos permite mejorar más aún si cabe el proceso de atención de mensajes de cada ventana

Posterior a esto, se puede observar el código generado para atender a los mensajes, donde resalta una zona delimitada por las macro BEGIN_MESSAGE_MAP y END_MESSAGE_MAP, que es donde se incluyen la lista de los mensajes a atender por la clase y su correspondiente función. Esta zona tampoco debe ser modificada manualmente sino que se actualiza con la utilización de ClassWizard o ClassBar.

Las zonas delimitadas como //AFX_... no deben ser editadas a mano, ya que son modificadas por Visual C++

CONCLUSIONES

En este artículo se a pretendido acercar al lector el método común de enlace del código con los recursos, creando funciones miembro para gestión de mensajes y clases nuevas para atender a los diálogos diseñados. El lector debe ir ampliando su aplicación siguiendo los pasos normales del diseño y que aún pareciendo a primera vista un tanto confusos, al modificar alegremente el código fuente, nos liberan de muchas horas de teclear código y de otras tantas horas de revisar los errores introducidos.

En el siguiente artículo se analizarán los diversos métodos principales de las clases *CApp* y *CWnd*, para poder ir particularizando más aún en las aplicaciones a desarrollar, bien como ejemplo, bien particularmente por parte del,lector.

LISTADO 2

```
CConfigDialog::CConfigDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CConfigDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CConfigDialog)

	//}}AFX_DATA_INIT
}

void CConfigDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CConfigDialog)

	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CConfigDialog, CDialog)
	//{{AFX_MSG_MAP(CConfigDialog)
	ON_BN_CLICKED(IDC_BOTON_PRUEBA, OnBotonPrueba)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CConfigDialog::OnBotonPrueba()
{
	// TODO: Add your control notification handler code here

	MessageBeep((UINT)-1);
}

void CConfigDialog::OnOK()
{
	// TODO: Add extra validation here

	MessageBeep((UINT)-1);
	CDialog::OnOK();
}
```

MEMORIA EXTENDIDA XMS

Enrique de Alarcón

El estándar XMS (eXtended Memory Specification) está destinado a la gestión de la memoria física situada por encima de la memoria convencional , que llega tan sólo hasta los 640 Kbytes. Pudiendo controlar así toda la memoria que esté disponible por encima de ésta , incluyendo la memoria UMB entre los 640-1024Kbytes y la HMA situada entre 1024-1088Kb. Es un driver que viene actualmente incluido en el sistema operativo MS-DOS. Este estándar fue creado hace ya años por las compañías Microsoft , Lotus , Intel y Ast , tres de las cuales, recordemos, fueron las creadoras también del anterior estándar LIM-EMS para la gestión de memoria Expandida , como se explicó en artículo del número de Febrero.

Actualmente , el controlador HIMEM.SYS de memoria XMS va ya por la versión 3.0 y todos los programas que no usan el moderno gestor DOS4GW de la casa Watcom o algún equivalente (que por cierto, está actualmente de moda), usan este driver. En el presente artículo se explica todo sobre su funcionamiento así como una detallada explicación de las 18 funciones más usadas del driver, con lo cual, el programador ya tendrá toda la información para usarlo en sus propias aplicaciones.

ORÍGENES DEL GESTOR HIMEM.SYS

La primera vez que apareció del controlador HIMEM.SYS, fue cuando se presentó la versión 3.0 de Windows, el cual, disponía de tres modos de funcionamiento, que eran: REAL, ESTÁNDAR Y MEJORADO). La capacidad de estos dos últimos modos de poder usar toda la memoria disponible en el sistema, vino dada gracias al gestor HIMEM que daba al sistema operativo las funciones necesarias pare ello. A partir de ése momento, cada vez fueron apareciendo más aplicaciones que usaban la memoria XMS y no la EMS, llegando un momento en que la XMS se convirtió en el estándar absoluto en gestión de memoria. Una prueba de su importancia como estándar, es que el HIMEM.SYS viene ya en el MS-DOS desde hace años como componente importante del mismo, ya que una versión DOS sin controlador XMS, no tendría ninguna salida comercial.

PRIMER PASO

Al igual que pasaba con el driver de memoria EMS, el gestor XMS no es una librería a linkar con las aplicaciones que realizemos durante su montaje, sino como se ha explicado anteriormente, un driver proporcionado por el propio sistema operativo que debe cargarse dentro del CONFIG.SYS como tal. Para ello debemos insertar la línea DEVICE=C:\DOS\HIMEM.SYS antes de la carga del gestor de memoria EMS, dado que el gestor de EMS, lo que hace es simular memoria EMS mediante memoria XMS, por lo tanto, si no está anteriormente la XMS cargada, no podremos crear la EMS.

Este driver acepta muchos parámetros, los cuales son los siguientes:

/A20CONTROL: ON/OFF .> Activa o desactiva la patilla A20 del micro que controla la memoria HMA. Por defecto está en ON.

/CLUCLOCK: ON/OFF > Si el ordenador pierde velocidad de proceso al usar himem, debe activarse esta opción para normalizar el funcionamiento del ordenador. Por defecto está en OFF.

/EISA > Si es un ordenador con más de 16Mbytes de memoria XMS, debe usarse para que el gestor use toda la memoria.

/HMAIN> Determina al gestor cuanta memoria HMA debe requerir una aplicación para que éste le permita asignar el bloque.



Con la aparición de los micros 286/386, que pueden gestionar hasta 4Gbytes de memoria física lineal en modo protegido, nació la necesidad de poder hacer que las aplicaciones DOS en modo real pudieran acceder de forma eficiente a esta memoria por encima del primer Megabyte mediante algún tipo de driver, apareciendo así el estándar XMS

/INT15=xxx > Determina la memoria que usa el interface de la interrupción 15h para su uso. Valores entre 0 y 63.
/NUMHANDLES=n > Número de bloques de memoria XMS (o handles) que pueden haber activos simultáneamente.
/MACHINE = n > Si es un modelo de ordenador especial no estándar , como algunos IBM's u otros , debe indicarse el modelo. La lista esta más al final de los parámetros.
/ SHADOWRAM: ON/OFF > Activa o desactiva la memoria Shadow Ram del sistema.
/TESTMEM: ON/OFF > Aciva o desactiva el chequeo de fiabilidad que realiza el driver al cargarse
de toda la memoria XMS para detectar SIMMS defectuosos o no fiables. Por defecto está en ON.
Lista de número y modelo de ordenador correspondiente:
1-IBM AT y 100%compatibles.
2-BM PS/2.
3-Phoenix Cascade BIOS.
4-HP Vectra.
5-AT&T6300 Plus
6-Acer 1100.
7-Toshiba 1600 & 1200x6.
8-Wise 12.5Mhz 286.
9-TulipSX.
10-Zenith ZBIOS.
11-IBM PC/AT.
12-IBM PC/AT
13-Philips.
14-Hp-Vectra
15-Equipo industrial IBM 77552.
16-Bell Micral 60.
17-Dell XBIOS.

LA MEMORIA UMB y HMA

Como ya se ha mencionado, el driver además de la XMS, gestiona también las llamadas memorias UMB y HMA.

La UMB (Upper Memory Block), es la memoria que está entre los 640Kbytes y los 1024Kbytes, que normalmente no está disponible para el DOS y que está destinada en principio para la ROM BIOS, tarjetas de vídeo, controladoras de disco y otros. Gracias al driver, el sistema encuentra las zonas de dicha memoria no usadas por el hardware, pudiendo así asignar bloques de memoria dentro de dicha zona igual que si creáramos un bloque de memoria DOS normal (con segmento en modo real, donde SEG*16= Dirección física).

La HMA (High Memory Area) son los 64Kbytes que se encuentran entre los rangos de direcciones 1024 y 1088 , y que se direccionan mediante la patilla A20 del procesador, que normalmente, está desactivada por cuestiones de compatibilidad con los viejos ordenadores. Gracias al driver, esta patilla puede activarse y usarse la memoria que contiene accesible directamente por el DOS.

CÓMO FUNCIONA LA MEMORIA XMS

Cada vez que asignemos un bloque de memoria XMS, el cual se llama técnicamente Bloque de memoria EMB (del inglés Extended Memory Bocks) el gestor nos devolverá un *handle* de 16 bits al estilo de los que dan las funciones de memoria DOS al asignar memoria convencional, aun que en este caso, el *handle* no sirve como segmento, si no que se usa como identificador del bloque XMS asignado. A partir de ahí , cada vez que queramos realizar alguna operación sobre el bloque creado, deberemos indicar el Handle correspondiente para que el gestor sepa de que bloque se trata.

Otra peculiaridad es que los bloques de memoria deben protegerse una vez asignados, ya que debemos recordar que este gestor está pensado para funcionar como base en sistemas multitarea como Windows. Como podemos deducir, al estar en modo real, no podemos manipular directamente la memoria XMS ya que está fuera del alcance del rango direcciones físicas accesibles desde nuestras aplicaciones. Por eso, debemos copiar los bloques de memoria XMS dentro de la zona de memoria convencional y viceversa mediante servicios del driver antes de poder manipularlos, el cual si tiene acceso a este rango de direcciones (ya explicaremos como). Este funcionamiento podemos compararlo a grandes rasgos con el sistema de gestión de la memoria EMS, pero sin la limitación de un marco de página de posición y tamaño fijo, y sin que se actualizen automáticamente los cambios realizados dentro de la memoria convencional en los bloques originales XMS.

CÓMO FUNCIONA EL GESTOR

Como ya se ha comentado, la aplicación DOS no tiene acceso directo a la memoria XMS, pero el gestor HIMEM sí. ¿Cómo se entiende esto? pues es muy fácil, lo que realiza en realidad el gestor al cargarse en memoria, es poner la CPU en modo real virtual, haciendo que funcione como un ordenador 8086 virtual dentro de un sistema mayor que funciona en modo protegido. Esa imposibilidad de acceder directamente a la XMS es la razón de que tengamos que usar servicios del gestor para poder copiar bloques XMS en convencional y viceversa.

Debido a esta forma de funcionar del ordenador, más adelante, en este mismo artículo, se explicará una método desconocido por muchos programadores de poder acceder directamente a la memoria extendida desde nuestras aplicaciones mediante la ampliación del rango de direcciones físicas direccionables en modo real (!).

CÓMO USAR EL GESTOR

El gestor de memoria XMS no se le llama mediante una interrupción como se hacía con el gestor EMS, si no que se accede a sus servicios mediante llamadas CALL dword ptr ... directas hasta el inicio del código del driver, donde hay una rutina principal del gestor que detecta el servicio que le pedimos dentro de AH y ejecuta la parte de su código correspondiente. La dirección donde debemos saltar, la debemos encontrar mediante un servicio de interrupción. Los parámetros se indican en los registros como si a servicios de interrupción llamáramos. Así pues, en AH indicaremos siempre el número de servicio que queremos usar , y en AX nos devolverá también los errores (si es que el servicio puede dar errores).

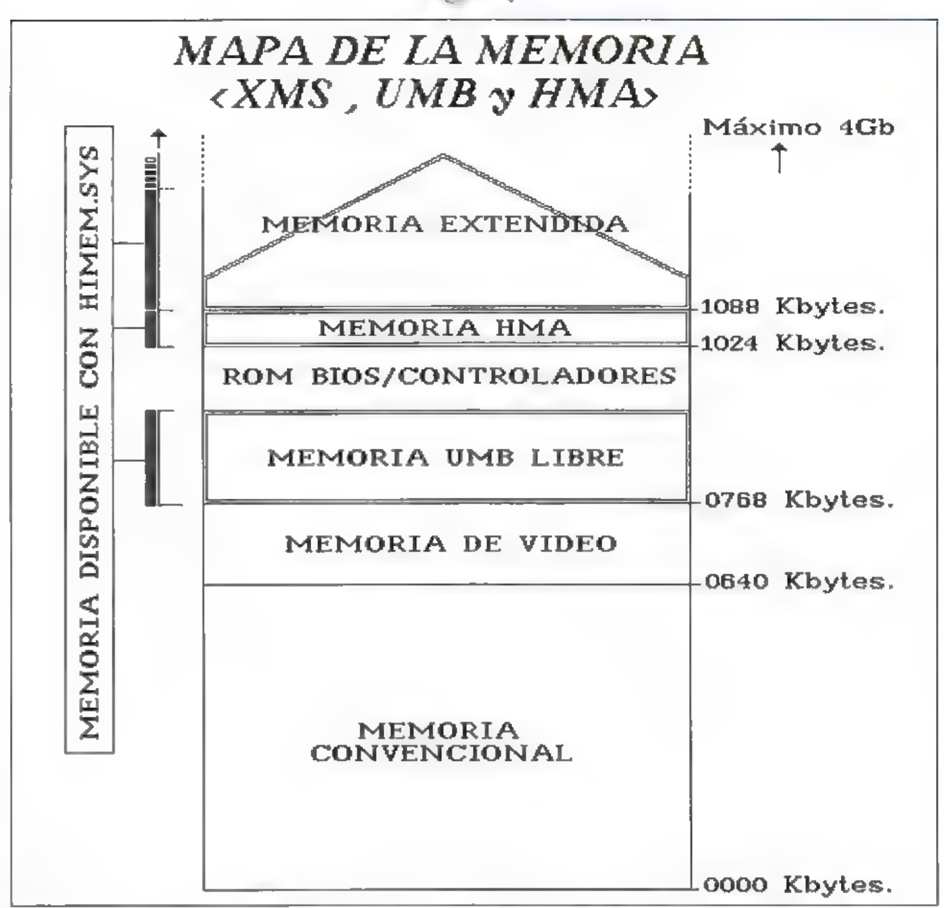
CÓMO INICIALIZAR EL GESTOR

Antes de poder usar ningún servicio del gestor XMS, debemos saber si está cargado o no en memoria y después obtener la dirección de salto que debemos indicar en las llamadas CALL para ejecutarlo. Para saber si un driver XMS está cargado en la memoria (que no debe ser forzosamente el HIMEM.SYS del que hablamos) , debemos ejecutar la interrupción 2Fh con el valor AH=43h y en AL el valor 00H, al ejecutar el servicio, nos devolverá , en caso de que esté el driver XMS, el valor 80h en AL.

Una vez hemos obtenido el valor AL=80h, debemos obtener la dirección

de llamada al driver, la cual obtendremos llamando de nuevo la interrupción 2Fh con el valor 4310h en AX. Está función, nos devolverá la dirección como SEGMENTO:OFFSET en los registros ES:BX. El código de inicialización del Driver XMS, quedaría como sigue:

```
INICIALIZA_XMS proc
  mov ah,43h
  mov al,00h
  int 2Fh
  cmp al,80h
  jne INIT_XMS_END
  mov ax,4310h
  int 2Fh
  push es
  pop XMS_SEG
  mov XMS_OFF,bx
  mov ah,00h
  call dword ptr XMS_SEG
  mov XMS_VERSION,ax
  mov CTR_VERSION,bx
  INIT_XMS_END:
  ret
INICIALIZA_XMS endp
```



Esquema de los 3 tipos de memoria disponibles con HIMEM.SYS.

FUNCIONES DISPONIBLES

A continuación se explica todas las funciones estándar de que dispone el gestor XMS. Recordemos que todas las funciones se llaman con el número de servicio en AH y que si han de devolver estados de error, lo harán en AX , con el valor 0000h si es error.

Función 00h: Obtener el número de versión del controlador XMS cargado en forma de número BCD de 16bits. Salida: AX=Número de versión BCD, BX=Versión del controlador, DX=0001h si hay memoria HMA presente, 0000h si no hay memoria HMA.

Función 01h: Reservar memoria HMA. Entrada DX=Espacio que queremos asignar (máximo 64 Kbytes). Salida: AX=0001h si se asignó el bloque HMA para nuestra aplicación, 0000h si no se pudo localizar.

Función 02h: Liberar el bloque de memoria HMA. Salida AX=0001h si se libera el bloque HMA anteriormente asignado. 0000h si no se libera.

Función 03h: Activar la patilla A20 a nivel global del SO. Salida AX=0001h si se pudo activar , 000h si no se pudo activar la patilla.

Función 04: Desactivar la patilla A20 a nivel global del SO. Salida: AX=0001h si se desactivo , 0000h si no se desactivó.

Función 05h: Activar la patilla A20 sólo a nivel local (de aplicación). Salida: AX=0001h si se activo localmente. 0000h si no se activó.

Función 06h: Desactivar la patilla A20 activada anteriormente a nivel local (de aplicación). Salida: AX=0001h si se desactivo localmente. 0000h si no se desactivó.

Función 07h: Obtener estado actual de la patilla A20. Salida: AX=0001h si está activada , 0000h si no lo está.

Función 08h: Obtener el tamaño del bloque de memoria XMS más grande que se puede crear actualmente y el número total de XMS instalada. Salida: AX=Tamaño en Kbytes del bloque más grande del que podemos disponer , DX=Tamaño total en Kbytes de memoria XMS que hay en el sistema.

Función 09h: Asignar un bloque de memoria XMS. Entrada: DX=Tamaño del bloque que queremos asignar en Kbytes. Salida: AX=0001h si se pudo localizar un bloque de dicho tamaño , DX=Handle del bloque de memoria que hemos asignado para nuestra aplicación.

Función Ah: Liberar un bloque de memoria XMS asignado anteriormente. Entrada: DX=Handle correspondiente a dicho bloque de memoria. Salida: AX=0001h si se pudo liberar , AX=0000h si no se pudo liberar.

Función Bh: Copiar un bloque de memoria XMS desde XMS a convencional , o desde XMS a XMS o de convencional a XMS. Entrada: DS:SI=Puntero a la tabla de parámetros de entrada (por su cantidad no se indican mediante registros directos). Salida: AX=0001 si se copió el bloque , 0000h si no se copió.

La tabla de parámetros incluye lo siguiente:

Offset 0h-3h = Número 32 bits con los bytes de memoria a copiar.

Offset 4h-5h = Número 16bits con el *handle* origen del bloque (0000h indica que bloque origen es de memoria convencional).

Offset 6h-9h = Offset 32bits del bloque origen (o SEGMENTO:OFFSET normal si es bloque de memoria convencional).

Offset Ah-Bh= Handle del bloque destino (0000h indica que destino es una zona de memoria convencional).

Offset Ch-Fh= offset 32bits al bloque destino (dirección SEGMENTO:OFFSET real normal si destino es bloque de memoria convencional).

Función Ch: Proteger un bloque de memoria XMS anteriormente asignado. Recordar que el XMS puede gestionar entornos de multitarea , como WINDOWS , para el cual fue creado. Entrada: DX=Handle del bloque XMS que queremos proteger. Salida: AX=0001h si se pudo proteger, 0000h si no se realizó la protección. DX=OFfset 32bits al inicio del bloque XMS que hemos protegido.

Función Dh: Desproteger un bloque de memoria XMS. Entrada: DX=Handle del bloque de memoria XMS a desproteger. Salida: AX=0001h si se desprotegió , 0000h si no se desprotegió.

Función Eh: Obtener información de un bloque XMS que tengamos asignado. Entrada DX=*handle* del bloque del que queremos obtener información. Salida AX= 0001h si se encontró información asociada , BH=Estado de protección del bloque XMS , BL=*handles* libres del sistema , DX=Longitud del bloque en Kbytes.

Función Fh: Modificar tamaño de un bloque de memoria XMS. Entrada: BX=Nuevo tamaño del bloque en Kbytes , DX=*handle* del bloque XMS que queremos modificar. Recordar que para poder modificarlo no puede estar protegido. Salida AX=0001h si se modificó el tamaño , AX=0000h si no se modificó.

Función 10h: Asignar bloque de memoria Superior. Entrada: DX=Tamaño en párrafos del bloque que queremos asignar (bloques de 16bytes) igual que si fuese un bloque de convencional. Salida: AX=0001h si se asignó , BX=Segmento del bloque asignado , DX=Tamaño del bloque asignado (en párrafos) o tamaño máximo que podemos asignar (en caso de error y AX=0000h).

Función 11h: Liberar un bloque de memoria superior. Entrada DX=Segmento del bloque UMB a liberar. Salida: AX=0001 si se liberó , 0000h si no se liberó.

EJEMPLO DE FUNCIONAMIENTO

Supongamos que queremos realizar una programa que cargue en memoria extendida un gráfico de 320*200 , lo dibuje en pantalla y que después lo libere. El código debería seguir los siguientes pasos:

1-Detectar si está el driver cargado. (INT 2Fh , AX=4300h).
2-Localizar la ubicación de memoria del Driver. (INT 2Fh ,AX=4310h).
3-Asignar un bloque EMB de memoria extendida de 64000 Bytes.
4-Asignar un bloque de memoria convencional de 64000 bytes.
5-Cargar el fichero con los 64000 bytes de gráfico en memoria convencional.
6-Copiar el bloque en el EMB asignado.
7-Liberar la memoria convencional.
8-Borrar la pantalla y copiar el bloque XMS en pantalla.
9-Liberar el bloque EMB.
10-Salir normalmente del programa.

SÓLO PARA EXPERTOS

Como hemos dicho anteriormemte, hay un detalle de los 386 y superiores que no se explica nunca, y que puede ser de interés para muchos programadores. Se trata de que podemos en realidad acceder a toda la memoria desde modo real.

Los 386, cuando funcionan en modo real, realmente están regidos mediante una tabla de descriptores igual que si estuviese en modo protegido. Dicho esto, se puede deducir, que lo que hace el 386 más o menos es simular un 8086 creando simplemente una tabla de descriptores con las base origen en el offset 0h y con una longitud de 1024Kbytes.

Dada esta información, se puede a su vez, deducir que, pasando a modo protegido, cambiando todos los límites de los segmento definidos en la Tabla de descriptor global a 4Gbytes, reactualizando todos los selectores de nuevo y volviendo a modo real sin modificar dicha tabla, tendremos a partir de ese momento, un ordenador funcionando en modo 8086 real que tiene acceso a toda la memoria disponible del sistema, con todos los segmentos de la GDT con base 0 y con longitud 4Gbytes.

Después, como las funciones del gestor XMS nos dan los offsets 32 bits en relación a base 0 de todos los bloques XMS que asignamos cuando los protegemos, tenemos ya punteros listos para acceder a dichos bloques directamente sin tener que depender de las engorrosas funciones de copia de bloques mediante tablas de parámetros que nos da el Driver HIMEM.

El único problema de lo explicado es que no se podrá realizar dicho paso de modo REAL a PROTEGIDO en ordenadores que tengan cargado un controlador EMM386, ya que dicho driver bloquea los derechos de privilegio de la CPU, dejando el nivel de privilegio 0, para él mismo, como si fuese un sistema operativo y dando un error de violación de acceso al intentar acceder desde niveles inferiores (o sea, desde nuestra aplicación). Tampoco podremos realizarlo en aplicaciones DOS funcionando bajo el S.O. Windows, ya que el mismo, también se asigna los derechos de privilegio de nivel 0, dejando a las aplicaciones si ningún tipo de acceso. Es esquema del código, quedaría como sigue:

1-Si se detecta EMS, fin proceso.
2-Copia la nueva Global Descriptor Table (con bases y límites maximizados) en la GDT
3-Pasa a modo protegido (Activa bit 0 del registro CR0 de la CPU).
4-Asigna los nuevos valores de la GDT a todos los segmentos reactualizando sus valores (MOV DS,BX MOV ES,BX...etc...)
5-Vuelve al modo real (desactivando el bit 0 de nuevo).

ÚLTIMOS DATOS DE INTERÉS

El bloque de memoria HMA que se ha comentado, pone a disposición de nuestras aplicaciones el gestor, no debe considerarse como algo disponible cuando creemos nuestras aplicaciones, ya que por ejemplo, cuando se ejecute nuestra aplicación bajo Windows, la cual puede tener varias aplicaciones funcionando simultáneamente, puede ser que dicho bloque esté asignado a otra aplicación. Así mismo, la memoria UMH, tampoco debe considerarse mucho, ya que normalmente el sistema operativo ya la usa prácticamente toda cargando drivers en ella. (las famosas funciones LOADHIGH= del DOS).

RESUMEN

En este artículo, se ha dado toda la información técnica sobre el estándar de memoria XMS, explicando tanto su origen , los tipos de memoria que gestiona, cómo se inicializa el driver y cómo se accede a él. También se han dado las referencias de todas las funciones que incluye, con lo cual, el lector está en posición ya de poder usar dichas funciones en sus propias aplicaciones.

CONTENIDO DEL CD-ROM

En el CD-ROM de este mes se ha incluido una recopilación de los principales libros y documentos de libre distribución acerca de linux, accesibles a través de Internet bajo el nombre de *La Biblia del Linux*.

LA BIBLIA DEL LINUX

La Biblia del linux de Sólo Programadores es una adaptación de *The Linux Bible 3rd Edition* de Yggdrasil preparada para ser consultada desde DOS. Contiene el grueso de la documentación existente sobre Linux en Internet: los principales libros, la mayoría de los *HOWTO*, *info pages* y una versión del *Linux Software Map (LSM)*. La versión original se sustentaba sobre nombres largos y la mayoría de los enlaces no funcionaban en DOS. Para hacerla accesible desde cualquier sistema ha sido necesario modificar los enlaces internos de más de 2.000 ficheros.

La totalidad de los trabajos se encuentra disponible en formato HTML y se puede consultar desde cualquier browser (cliente) de Web que permita acceder a ficheros locales, como Netscape, Mosaic, WinWeb o Lynx. Para ello, bastará abrir desde el browser el documento *top.htm* situado en la raíz de la Biblia del Linux. En este documento, encontrará enlaces a los distintos tipos de información almacenada y podrá consultarlos cómodamente como si navegara a través de Internet. Si en determinado momento desea salir de un libro y entrar en otro, deberá acceder nuevamente al fichero local *top.htm*. No se garantiza el correcto acceso a una determinada página si no es desde *top.htm*.

Asimismo, se encuentran las fuentes de los documentos en muchos otros formatos, como PostScript o SGML.

Nota Importante: Para utilizar NetScape sin soporte de red, es necesario copiar el fichero MOZOCK.DLL, quese encuentra en el directorio BROWSERS, al directorio WINDOWS con el nombre WINSOCK.DLL.

LIBROS INCLUIDOS

Los libros incluidos en la presente recopilación comprenden:

Installation and Getting Started de Matt Welsh

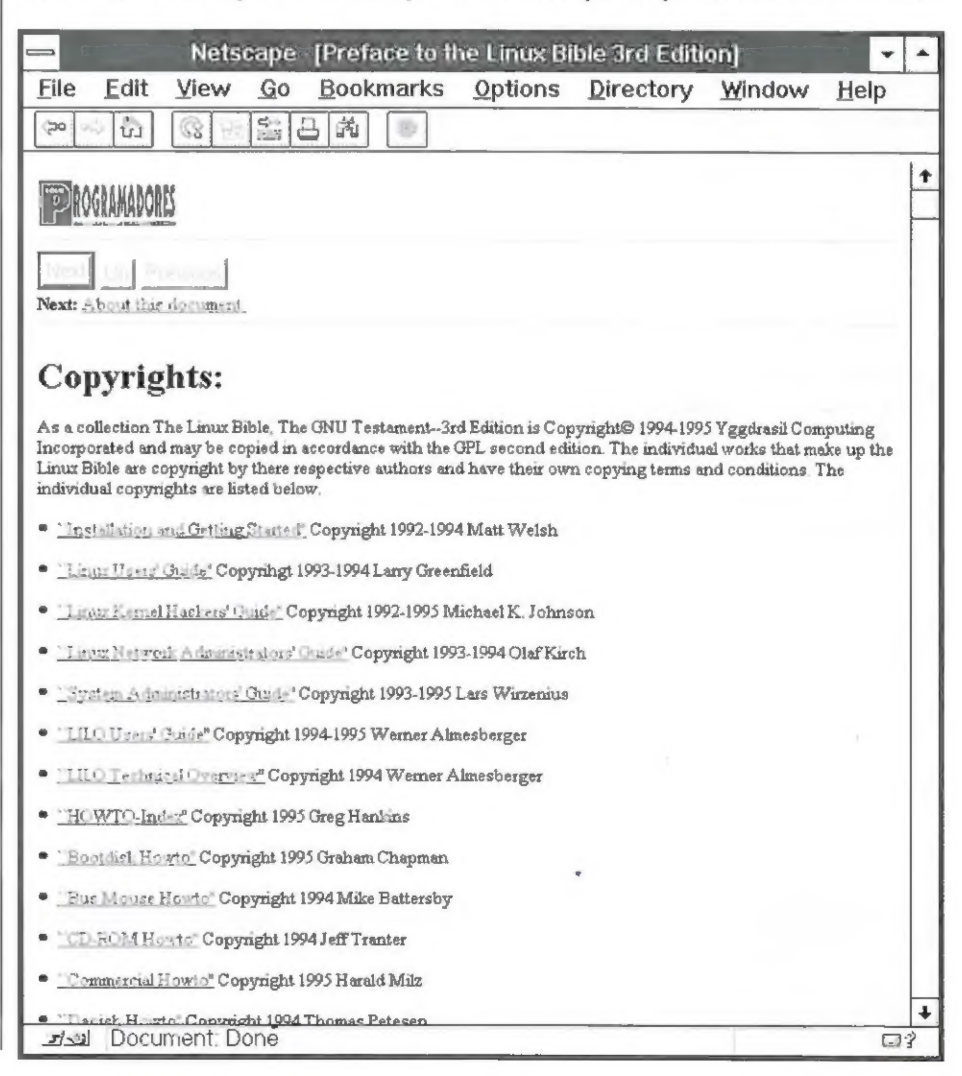
Una guía de instalación de Linux y referencia para nuevos usuarios de UNIX. La información que contiene comprende desde cómo realizar una instalación de cualquier distribución de Linux, a cómo administrarla, pasando por un breve curso de UNIX para aquellos usuarios que entren en contacto por primera vez con este sistema.

Linux Users' Guide de Larry Greenfield

Otro libro de referencia que se centra sobre todo en los aspectos de manejo de UNIX a nivel de usuario, explicando desde las órdenes más simples, el uso de las shells, el sistema de ventanas X-Window, el editor vi, algunos programas de aplicación, configuración y personalización, comunicaciones con otros usuarios, etc. En definitiva, una de las mejores formas de aprender UNIX a nivel de usuario.

Linux Kernel Hackers' Guide de Michael K. Johnson

Un libro denso y profundo para aquellos que deseen conocer el funcionamiento interno del núcleo de Linux. Este libro se encuentra en constante evolución debido a que el kernel de Linux también lo está, por lo que debe servir como refe-



rencia únicamente y, si es posible, acceder siempre a la última versión disponible en Internet. Cubre aspectos como los drives de dispositivos, el *filesystem /proc*, el gestor de tareas, funcionamiento de llamadas al sistema, programación de drivers SCSI y gestión de memoria.

Linux Network Administrators' Guide de Olaf Kirch

La referencia indispensable para aquellos usuarios que dispongan de una máquina Linux conectada a una red o que deseen aprender acerca de administración de redes. Aunque el libro esta dedicado a Linux, se explican claramente todos los conceptos necesarios para la administración de cualquier sistema UNIX en red.

System Administrators' Guide de Lars Wirzenius

Un compendio de información, aún en desarrollo, sobre temas referentes a la administración de sistemas Linux. El lugar de consulta idóneo una vez se ha terminado con el *Installation and Getting Started*. Trata temas diversos como el árbol de ficheros, inicialización y cierre del sistema, uso de dispositivos de almacenamiento, sistemas de ficheros, memoria virtual, backups y el sistema de ficheros propio de Linux, *ext2fs*.

LILO Users' Guide de Werner Almesberger

Todos los usuarios que instalan Linux, antes o después se ven las caras con *LILO*, el gestor de arranque. Aunque su configuración automática no suele dar problemas, a través de esta guía se explican todos sus entresijos y formas de personalización, así como su funcionamiento básico. Se incluye, además, una guía para la resolución de problemas.

LILO Technical Overview de Werner Almesberger Una guía para el programador que explica el funcionamiento interno de LILO.

HOWTOs

Los *HOWTOs* de Linux son documentos lo más claros, breves y concretos posible acerca de un determinado aspecto de la instalación, configuración o manejo de una determinada área de Linux. Son la primera referencia a consultar cuando se desea resolver un problema concreto, por ejemplo, la instalación de una impresora.

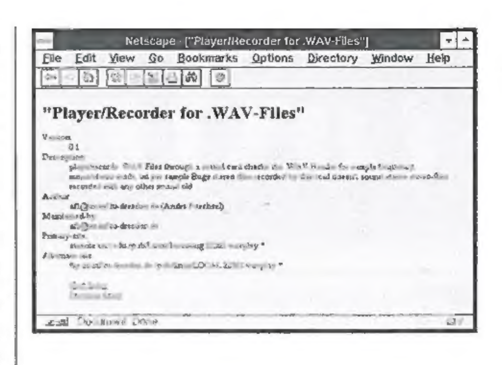
La Biblia del Linux de Sólo Programadores contiene los siguientes HOWTOs: *HOWTO-Index* de Greg Hankins, *Bootdisk Howto* de Graham Chapman, *Bus Mouse Howto* de Mike Battersby, *CD-ROM Howto* de Jeff Tranter, *Commercial Howto* de Harald Milz, *Danish Howto* de Thomas Petesen, *Device List* de H. Peter Avin, *DOS Emulator Howto* de Mike Deisher, *Distribution Howto* de Erik Troan, *ELF Howto* de Riccardo Facchetti, *Ethernet Howto* de Paul Gortmaker, *Firewall Howto* de David Rudder, *Ftape Howto* de Kai Harrekilde-Petersen, *Ham Howto* de Terry Dawson, *Hardware Howto* de FRiC, *Installation Howto* de Matt Welsh, *JE Howto* de Hirondu ABE y Yasu Hiro, *Kernel Howto* de Briam Ward, etc.

LINUX SOFTWARE MAP (LSM)

LSM es un intento de recopilar información sobre todo el software existente para Linux. Aunque la guía incluída en el CD-ROM ha quedado algo anticuada, aún es un estupendo marco de referencia para conocer de que software se puede disponer para Linux y cómo conseguirlo.

INFO PAGES

Las *info pages* contienen información dirigida al usuario más avanzado, cubriendo desde breves documentos hasta completos manuales. En el CD-ROM se incluyen: 4.4 BSD Automounter Reference Manual, ange-ftp, GDB Annotations, Using as, Bash features, LIB BFD the Binary File Descriptor Library, The GNU Binary Utilities, Bison, On Configuring Development Tools, Common Lisp Extensions, Cygnus configure, The C Preprocessor, dc an arbitrary precision calculator, Comparing and Merging Files, Dired Extra Version 2 User's Manual, GNU ed version 0.2, GNU



Emacs Manual, Emu Reference Manual, Evi - Emulate/Enhanced vi for GNU Emacs, Fileutils, Flex, version 2.5, Forms Mode User's Manual, User's Guide to gperf, The GNU C++ Iostream Library, to the GNU C++ Library, GASP an assembly preprocessor, The GAWK Manual, Debugging with GDB, Using and Porting GNU CC, GDB Internals, GNU MP, Keeping Track, GNUS 3.14 Manual, GNU gprof, G++ internals, gzip, ILISP User Manual, Indent, Info 1.0, InterProcess Communication, International Ispell 3.1, Kpathsearch library, Using ld, GNU Linker Internals, Libgdb, GNU m4 version 1.4, GNU Make, Makeinfo, mmalloc, GNU Smalltalk User's Guide, Pcl-cvs - The Emacs Front-End to CVS, GNU recode version 3.4, GNU C++ Renovation Project, Supercite Version 3.1 User's Manual, Reporting Problems, GNU sharutils version 4.1, GNU Source Trees, The "stabs" debug format, Texinfo, Measuring Program Resource Use, Taylor UUCP, VIP, GNU wdiff version 0.5, Xemacs.

NOTA SOBRE COPYRIGHT Y LICENCIA

La presente recopilación de información, es una modificación de la Yggdrasil Linux Bible 3rd Edition bajo licencia GPL por lo que es de libre distribución y modificación. En la mayoría de los documentos que contiene se incluyen los términos de esta licencia. Cada uno de los trabajos individuales que contiene está sujeto a sus propios derechos de distribución y licencia, los cuales figuran claramente en el CD-ROM.

Esta conversión es una cortesía de Sólo Programadores hacia sus lectores, por lo que no se realizará ningún tipo de mantenimiento y/o actualización de la documentación aportada.

NOTA: Las páginas HTML del CD que hacen referencia a la biblia de Linux **sólo funcionan desde Windows 3.x**. Los usuarios de Windows 95 no podrán acceder a dichas páginas por problemas de incompatibilidad de formatos de ficheros y directorios entre Linux y MS-DOS.



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

P *Hola, soy un estudiante de informática de sistemas y en este momento estoy realizando el proyecto final de carrera y necesito la siguiente información:*
¿Existe algún tipo de DLL's que convierta una imagen almacenada en un formato gráfico en otro de los soportados por Visual Basic? También me valdrían módulos VBX que consiguieran los mismos resultados. ¿Donde puedo encontrar todo esto, en caso de existir?

Santiago Ibarra Jiménez
(Barcelona)

R Sí que existe lo que buscas. Desde luego lo mejor es recurrir a algún paquete comercial para Visual Basic especializado en proceso de imágenes, pero en su defecto puedes conseguir alguna de las innumerables librerías ya creadas que existen por todo el mundo. ¿Como conseguir algo en cualquier parte de mundo ? Pues mediante Internet, claro. Prueba a buscar en los siguientes *ftps*:
SimTel
ftp.coast.net
En el directorio:
/pub3/sintel_win3/visbasic

Microsoft
ftp.microsoft.com
En el directorio:
/Softlib/msfiles

P *Estoy estudiando la carrera de Ingeniería de Sistemas Computacionales y Electrónicos en la Universidad del Valle de Cochabamba (Bolivia), obtuve su revista en mis vacaciones por Argentina y me gustaría conocer como obtener números atrasados y/o suscribirme. Así mismo, quisiera pedirles que me ayuden con los siguientes temas:*
- ¿Como puedo trabajar con 256 colores o más desde el lenguaje Pascal, utilizando para ello las funciones gráficas habituales ?
- Podrían publican algún algoritmo de compresión de datos.
- ¿Como puedo crear mis propias fuentes desde Pascal, de manera que sean escalables como las True Type de Windows?
- Algo de información sobre ampliación y reducción de gráficos.
- ¿Que es más rápido: el acceso directo a la memoria de vídeo o realizarlo mediante la BIOS de la computadora?
Gracias por todo y espero me disculpen por la cantidad de preguntas, pues pasará mucho tiempo hasta que me llegue la respuesta.

Ángel Ventura Vaca Eyzaguirre
(Cochabamba / Bolivia)

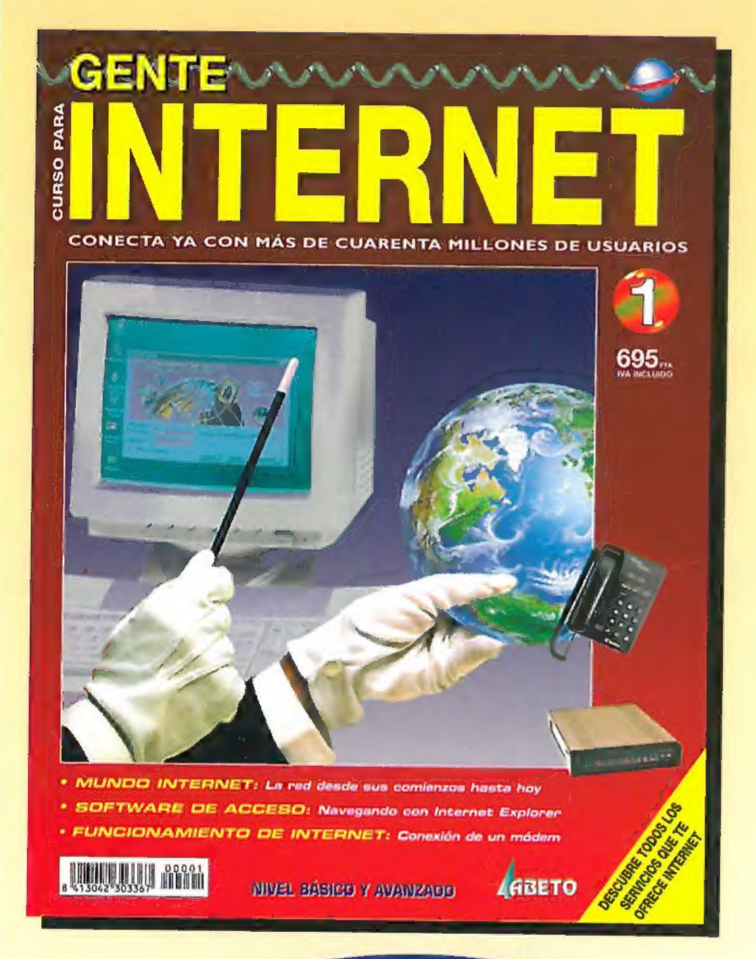
R Cada día somos más internacionales y parece que estamos consiguiendo un nutrido grupo de amigos al otro lado del Atlántico.
Para obtener cualquier número o suscribirse, hay que dirigirse a la siguiente dirección :
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
Pol. Ind. Alcobendas 28.100 (Madrid)
Fax: (91) 661 43 86
Teléfono: (91) 661 42 11

Para trabajar con 256 colores o más, tan solo hay que activar el modo gráfico correspondiente y escribir en la Vídeo-Ram los datos adecuados. Para poder realizarlo desde Pascal programando a alto nivel, es necesario disponer de las correspondientes librerías gráficas, que permitan utilizar las habituales funciones del tipo *rectangle, fill, draw*, etc. En las últimas versiones de los distintos compiladores de Pascal, ya se incluyen librerías para desarrollar, al menos, bajo 256 colores.

La manipulación de fuentes escalables es ardua y requiere un completo conocimientos de gráficos vectoriales. Básicamente, consiste en mantener la definición de "como" dibujar cada letra, no del dibujo propiamente dicho. Así se podrá plasmarla de la manera que se desee, incluso rotándola en cualquier ángulo.

El acceso a la memoria de vídeo siempre es mucho más rápido si se realiza directamente, pues el acceso a cualquier función de la BIOS ralentizará la ejecución por muchas razones : llamada a la función mediante una interrupción, la zona de memoria en donde se encuentra la BIOS (ROM) es muy lenta y las funciones son muy generales y muy poco optimizadas.
Sobre las preguntas acerca de información relativa al escalado de gráficos y algoritmos de compresión, lo mejor es buscar alguno de los artículos publicados en esta revista, pues son temas muy complejos.

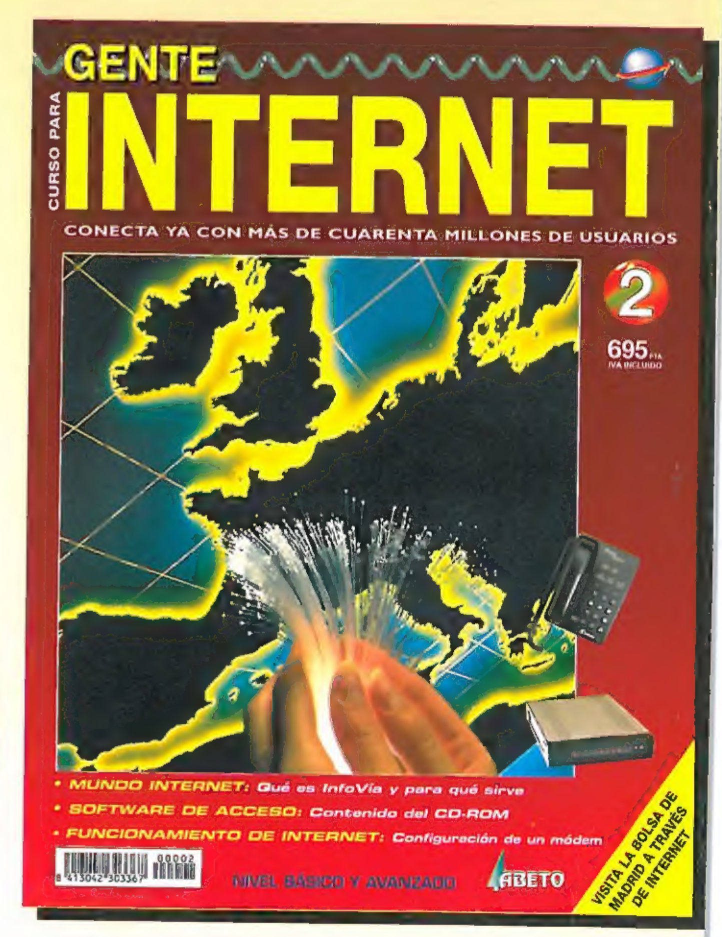
¡¡DOMINA YA INTERNET CON EL COLECCIONABLE GENTE INTERNET!!



15 DÍAS DE CONEXIÓN GRATIS







INTERNET ES PARA TODOS

- Abogados • Profesionales
- Empresarios • Periodistas
- Médicos • Ingenieros • Arquitectos
- Estudiantes • Economistas
- Usuarios en general...

TODAS LAS SEMANAS EN TU QUIOSCO

LA COLECCIÓN CONSTA DE 40 FASCÍCULOS, 39 DISQUETES, 1 CD-ROM Y 4 TAPAS



c/Aragoneses, 7 - Pol. Ind. Alcobendas - 28100 MADRID Tel.: (91) 661 42 11 - Fax (91) 661 43 86

ERVICIO 24 HOR
06 - 31 32